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

---

Etude de classes de fonctions de coût régulières

Thèse présentée pour l'obtention du diplôme de

Docteur de l'Université Paris Diderot, spécialité Informatique

à l'École Doctorale de Sciences Mathématiques de Paris Centre

Par

Denis Kuperberg

Thèse dirigée par Thomas Colcombet

Soutenue publiquement
le 10 décembre 2012
devant le jury constitué de :

Directeur de Thèse :	M. Thomas Colcombet
Rapporteurs :	M. Mikołaj Bojańczyk
	M. Igor Walukiewicz
Examinateurs :	M. Olivier Carton
	M. Christof Löding
	M. Jean-Eric Pin
	M. Philippe Schnoebelen

Remerciements

Je tiens tout d'abord à remercier mon directeur de thèse, Thomas Colcombet, sans qui ce document n'aurait jamais existé, et qui s'est vite révélé irremplaçable. Sa passion et sa vision de la recherche sont pour moi une source constante d'inspiration. Sans se contenter de transmettre du contenu scientifique, il a toujours pris le temps nécessaire pour me préparer au métier de chercheur de la meilleure manière possible, en partageant son expérience et ses conseils à chaque occasion. Sa joie et sa curiosité sont communicatives (les membres du labo pourront témoigner de son humour proverbial), et je garderai un très bon souvenir de toutes nos discussions, guidées par le plaisir de faire de la recherche. Je ne peux qu'espérer avoir le même enthousiasme durant ma carrière. Je le remercie également ainsi que sa femme Marion (et toute la troupe !) pour leur hospitalité fréquente malgré les nombreuses obligations familiales.

Merci à Mikołaj Bojańczyk et Igor Walukiewicz d'avoir accepté d'être rapporteurs, ainsi que pour leurs commentaires dans les rapports. J'apprécie à sa juste valeur la charge de travail qu'ils ont dû fournir pour rendre dans les temps des rapports sur ce document relativement long, malgré leur emploi du temps chargé. Merci également à Olivier Carton, Jean-Eric Pin, Christof Löding et Philippe Schnoebelen d'avoir accepté de faire partie du jury.

Une partie importante de cette thèse est le fruit d'un travail en collaboration avec Michael Vanden Boom, que je remercie chaleureusement. Nos nombreux échanges Oxford/Paris et tout le temps passé en sa compagnie constituent autant de bons souvenirs, et sa bonne humeur autant que ses qualités scientifiques ont fait de cette collaboration une expérience réussie. Je remercie également Nathanaël Fijalkow pour avoir relu et commenté des parties de ce manuscrit.

Le LIAFA a constitué un cadre de travail révé, et tous ses membres sont à remercier pour leur participation à l'ambiance stimulante et conviviale qui y règne. Je suis particulièrement reconnaissant envers Noëlle et Nathalie pour la patience dont elles ont fait preuve au cours de ces trois ans. Pouvoir compter sur leur soutien et sur leur sympathie a été une aide précieuse, sans laquelle ce travail aurait été impossible. Merci également à Olivier Carton et Jean-Eric Pin d'avoir souvent pris du temps pour m'épauler dans les labyrinthes administratifs que j'ai eu à affronter.

Merci aux membres de l'équipe FREC Paris : Jean-Eric, Olivier, Thomas, Mai, Laure, Charles, Luc, Nathanaël, Sam, et Yann, pour les croissants/exposés du lundi matin et les réunions dans le sud-ouest. Je salue au passage la composante FREC Junior, en espérant qu'elle a encore de nombreuses soirées vin/saucisson devant elle. Luc, Laure, Charles et Nathanaël ont tout spécialement fait partie du quotidien de ma thèse, et leur compagnie tant au travail que dans la détente a été un plaisir toujours renouvelé. Je ne remercierai jamais assez mes collègues de bureau : Thach, Anh, Xavier, Hervé, Mauricio, Heger, Charles, Yann et Nathanaël, grâce à qui les rires agrémentaient souvent les séances de travail. Ils sont pour beaucoup dans ma joie quotidienne de pousser la porte d'entrée du labo. Merci aussi à Jean, Thach, Irène, Cezara et Christophe, qui m'ont recueilli au sein du

fameux club piscine du LIAFA, pour de nombreuses escapades aquatiques. Plus généralement, merci à tous les thésards (et permanents) pour les innombrables repas dans la bonne humeur, les pause-café toujours enrichissantes, les pique-niques, les parties de go, les croissants du séminaire du mercredi, le foot, et la bonne humeur quotidienne.

En dehors de la vie professionnelle, j'adresse mes remerciements sans limite aux colocataires Dunois : Chep, Emilie, Fafa, Réjane, Sofiane, Mehdi, Carolina. Leur réserve inépuisable d'amitié a été un socle indispensable au cours de ces trois années (et j'espère pour de longues années encore), et la richesse des expériences vécues avec eux a été aussi épanouissante que rafraîchissante, ce qui n'a pas de prix quand on est en thèse. Merci tout spécialement à Réjane qui a toujours prêté une oreille attentive à mes péripéties professionnelles, et qui a fait preuve d'une curiosité scientifique jamais épuisée malgré la distance qui sépare nos deux domaines. Je ne ferai pas la liste des habitués des lieux qui ont contribué à mettre de la vie dans l'appartement et dans Paris, mais ils se reconnaitront... Merci à Florent pour sa régularité sans faille aux lundi enflammés à Notre-Dame, ainsi qu'aux divers festivaliers ou vacanciers (ils se reconnaitront aussi) qui n'ont pas eu peur de s'entasser dans des voitures douteuses pour partir en escapade à la première occasion. Je remercie également les amis de l'ENS Lyon pour les liens profonds qui ont été tissés durant ces années et pour tout ce qu'ils m'apportent encore aujourd'hui.

Merci enfin du fond du coeur à ma famille, qui m'a toujours apporté un soutien inconditionnel et précieux, tout en me poussant régulièrement à devenir quelqu'un de meilleur. Je remercie ma mère de m'avoir donné le goût des sciences et des mathématiques, et mon père de m'avoir fait découvrir l'informatique, et de m'aider à garder les pieds sur terre. Merci également à Claudius, Fabienne, Marie, Charlotte et Aurélien pour de nombreux souvenirs heureux, et pour leur enseignement qui a contribué à me construire. Je témoigne finalement mes amitiés les plus respectueuses à mon frère Benjamin, qui n'a jamais été découragé par la distance Paris-Grenoble, et avec qui j'ai partagé plus de fous rires que je ne pourrais en compter.

Table des matières

Introduction

Contexte

Du point de vue théorique, un grand nombre de questions se réduisent à étudier un ensemble de mots, appelé *langage*. Un langage peut être spécifié de plusieurs manières différentes : par exemple on peut définir le langage des mots comportant un nombre premier de lettres, ou celui des mots ne comportant que des 1. Cependant, on doit toujours trouver un moyen de décrire de manière finie un langage.La solution la plus naturelle est de décrire une propriété qui caractérise les mots du langage, comme on le fait dans les deux exemples précédents. On s'intéresse entre autres aux liens pouvant exister entre la manière de spécifier un langage, et les propriétés qu'aura ce langage. En particulier les *machines de Turing* permettent de décrire exactement (selon la thèse de Church-Turing) les langages caractérisés par une propriété de nature algorithmique. Les machines de Turing présentent cependant quelques inconvénients :

- Le modèle de calcul n'est pas réaliste car il dispose d'une mémoire non bornée.
- Beaucoup de questions naturelles, comme l'arrêt d'une machine sur une entrée donnée, sont *indécidables*[1].

Il est donc important d'étudier un modèle simplifié de machines de Turing, par exemple en bornant la taille de la mémoire. Ceci correspond au formalisme des *automates finis*, déjà utilisés implicitement dans plusieurs domaines (tels que les probabilités ou l'ingénierie) antérieurement aux machines de Turing. La classe des langages reconnus par de tels automates s'appelle la classe des *langages réguliers*, et est caractérisée par de nombreux autres formalismes, issus de domaine variés tels que la logique, l'algèbre, les expressions régulières. Cette classe semble donc renfermer une propriété *intrinsèque*, indépendante du modèle que l'on choisit pour spécifier des langages. La théorie des langages réguliers a été initiée dans les années 1950 par Kleene [Kle56], Rabin et Scott [RS59], elle a donné lieu depuis à des applications dans des domaines variés tels que la compilation, la vérification, l'algorithmique du texte, ou encore la linguistique.

Le développement de cette théorie a permis d'éclairer de nombreuses notions, au travers de connexions avec d'autres domaines comme l'algèbre ou la logique. Malgré ces nombreux liens mis en évidence, certaines questions sont toujours ouvertes, et nécessitent une meilleure compréhension de la théorie. Certaines de ces questions sont évoquées ici, par exemple le problème de la hauteur d'étoile généralisée, ou encore celui de la décidabilité de l'indice de Mostowksi non-déterministe (voir Section 13.3). D'autre part, cette théorie continue de s'étendre en incluant de nouveaux formalismes qui généralisent les langages réguliers.

La classe des langages réguliers présente de nombreux avantages :

- Elle est stable pour la plupart des opérations classiques sur les langages.
- Elle est caractérisée par des formalismes variés, que l'on peut utiliser suivant les besoins.

1. On a cependant besoin du modèle de la machine de Turing pour donner un sens précis à ce mot !

– De nombreux problèmes sont décidables sur cette classe, alors qu'ils deviennent indécidables pour des langages plus généraux.

Or, le modèle des mots finis n'est pas toujours le plus adapté : il peut être intéressant de s'intéresser à des structures plus diverses, pour décrire un ensemble de phénomènes plus riches. La notion de langages formels a été généralisée aux mots infinis, aux arbres (permettant de modéliser la notion de choix branchant), aux ordres linéaires, etc. Ceci conduit dans chaque cas à une généralisation de la classe des langages réguliers, avec l'espoir qu'un maximum des avantages décrits plus haut soient toujours présents.

Cependant, les propriétés que l'on peut exprimer au moyen de langages réguliers sont toujours de type booléennes. Un langage sépare simplement les « bonnes » structures des « mauvaises ». Il est naturel de vouloir nuancer ce critère d'acceptation, en affectant non pas un booléen à chaque structure, mais plutôt un nombre entier ou réel, ou même plus généralement un élément d'un ensemble fixé quelconque. On définirait ainsi non plus des langages, mais plutôt des *fonctions numériques* prenant des mots comme arguments.

Dans cette optique, de nombreux modèles quantitatifs d'automates ont été développés. On peut citer les *automates à poids* introduits dans [Sch61], encore très étudiés aujourd'hui, voir par exemple [DG07] pour de récents développements. Un cas particulier de cette notion est particulièrement naturel : les *automates probabilistes* [Rab63] pondèrent chaque transition par une probabilité d'être empruntée. Notons que le comportement d'un tel automate est celui d'une chaîne de Markov : toute l'information utile pour la prédiction des futurs possible est contenue dans l'état courant de l'automate. Finalement, les *automates temporisés* [LV92] permettent de situer les évènements dans le temps, et constituent un autre axe de généralisation quantitative des langages réguliers. Les formalismes logiques sont également étendus de manière quantitative dans une optique de vérification de systèmes formels. On pourra ainsi par exemple quantifier sur le nombre d'occurences d'un évènement [LMP10], leur importance [Boj04a], ou encore le temps qui les sépare [AH94, KPV09].

La théorie des *fonctions de coût régulières* participe à cette branche. Elle a été initiée par Colcombet [Col09], et généralise la théorie des langages réguliers de manière quantitative.

Cette théorie s'est développée suite à plusieurs branches de travaux antérieurs. Certains résultats ont été obtenus en théorie des langages réguliers en étudiant des automates munis de compteurs. Le problème de la hauteur d'étoile est emblématique, il pose la question de l'existence d'un algorithme répondant à la question suivante :

Entrée : Un langage régulier L et un entier k,

Sortie : « Oui » si L peut être représenté par une expression rationnelle utilisant au plus k étoiles de Kleene imbriquées, « Non » sinon.

Cette question a été posée par Eggan en 1963 [Egg63], mais résolue seulement 25 ans plus tard par Hashigushi au moyen d'une preuve très compliquée [Has82b, Has82a, Has83, Has88]. Une démonstration simplifiée a été présentée plus récemment par Kirsten [Kir05]. Les deux preuves utilisent le même principe : réduire le problème original à l'existence d'une borne pour une fonction à

valeurs entières, représentée au moyen d'un automate à compteurs (« distance automaton » pour Hashiguchi et « nested distance desert automaton » pour Kirsten). Ils ont ensuite montré que ce dernier problème est décidable. Il est à noter que les expressions rationnelles dont l'on parle ici ne comportent pas de complémentation (ni d'intersection). Le problème dit de « hauteur d'étoile généralisée » où la complémentation est autorisée n'est pas résolu à ce jour.

D'autres problèmes de décision se réduisent également à des questions de bornes pour ce type d'automates. On peut citer en théorie des langages le problème de la puissance finie [Sim78, Has79], et celui de la substitution finie [Bal04, Kir04]. De tels automates sont aussi utilisés dans d'autres contextes, comme la théorie des bases de données [GT07], la compression d'images [CK93], la reconnaissance de langues naturelles [Moh05], ainsi qu'en logique [BOW09] et en vérification [AKY08]. La théorie des fonctions de coût vise à placer ces techniques dans un cadre théorique plus général, avec l'espoir que son développement permettra de résoudre d'autres problèmes (issus par exemple de la théorie des langages). Des efforts dans ce sens ont déjà été effectués par Colcombet et Löding [CL08] dans le cas des arbres infinis, on détaillera cette contribution dans la Section 13.3.

Le développement de cette théorie a aussi bénéficié de travaux menés par Bojańczyk et Colcombet [Boj04b, BC06] portant sur des extensions de la logique monadique du second ordre (MSO) au moyen d'un quantificateur $\mathbb{U}$ spécifiant des propriétés sur la taille d'un ensemble. Ces travaux ont ensuite été poursuivis par Bojańczyk et Toruńczyk dans le cadre de la logique MSO faible [BT09, BT12].

A la lumière de toutes ces considérations, Colcombet [Col09] introduit la notion de fonction de coût, fournissant notamment de nombreux formalismes équivalents permettant de les définir : deux formes duales d'automates de coût (les B- et les S-automates), une extension de la notion de monoïde (les *monoïdes de stabilisation*), et une extension de la logique monadique du second ordre. La théorie des fonctions de coût ainsi construite forme une extension stricte de la théorie des langages, et développe de manière plus générale les notions introduites pour résoudre certains problèmes sur les langages classiques, dont par exemple celui de la hauteur d'étoile mentionné plus haut. La théorie des fonctions de coût a également été étudiée par Toruńczyk dans sa thèse [Tor11] au moyen d'outils topologiques. Il montre qu'il est possible de considérer les fonctions de coût comme des langages de mots profinis, et de prouver certains théorèmes dans ce cadre.

Contributions

L'une des premières contributions de la thèse est d'ordre théorique : il s'agit de généraliser la congruence syntactique définie sur les langages. Rappelons que pour tout langage régulier L, il existe un objet algébrique fini minimal décrivant L, appelé son monoïde syntaxique, et noté $\mathbf{S}_L$. De plus, on peut définir $\mathbf{S}_L$ simplement, en quotientant l'ensemble des mots $\mathbb{A}^*$ par une relation d'équivalence $\sim_L$ définie à partir de L. On a montré dans la thèse l'existence d'un monoïde de stabilisation minimal $\mathbf{S}_f$ pour toute fonction de coût régulière

f, et donné un algorithme pour l'obtenir à partir de toute description de f. On montre également qu'il est possible de définir une congruence syntaxique sur les fonctions de coût, permettant de décrire le monoïde de stabilisation minimal d'une fonction de coût comme un quotient. L'ensemble destiné à être quotienté dans le cas des langages était l'ensemble des mots finis $\mathbb{A}^*$, on utilise ici une extension de cet ensemble, en remplaçant les mots finis par la notion de $\sharp$-expression. Une première version de $\sharp$-expression a été introduite dans [Has90], que l'on généralise ici en $\omega\sharp$-expression.

Les autres contributions consistent en des caractérisations multiples de sous-classes restreintes. La première classe étudiée est celle des fonctions temporelles. L'idée qui sous-tend cette classe est la mesure du temps : les fonctions temporelles sont celles qui sont limitées à mesurer des intervalles d'évènements consécutifs (il est par exemple interdit de compter le nombre d'occurrences d'une lettre). On obtient plusieurs caractérisations équivalentes de cette classe : en termes d'automates de coût, en termes de langages réguliers (appelés *langages métronomes*), et en termes de monoïdes de stabilisation. La seconde classe étudiée sur mots finis est celle des fonctions définissables par CLTL, une extension quantitative de LTL, introduite dans cette thèse. On donne des traductions de CLTL vers les deux formes duales d'automates de coût, ainsi qu'un algorithme PSPACE permettant de décider si une formule est satisfaisable de manière bornée. Cette classe généralise celle des langages apériodiques, et on s'attache donc à obtenir un théorème analogue à celui de Schützenberger-McNauhgton-Papert, qui donne une équivalence entre un formalisme logique (CLTL ici, qui généralise LTL) et une condition algébrique d'apériodicité. Ceci permet d'obtenir la décidabilité du problème de l'appartenance à cette classe de fonctions de coût, pour toute fonction de coût régulière. Des caractérisations de cette classe par une logique de coût du premier ordre, ainsi que par un modèle d'automates alternants, sont obtenus dans le cadre plus général des mots infinis. Ces résultats généralisent ceux obtenus pour les langages classiques, comme le théorème de Kamp [Kam68]. On peut se reporter à [DG08] pour une présentation globale de ces résultats.

On caractérise l'intersection des deux classes ci-dessus, comme étant égale à la classe des fonctions de coût définissables par des formules de Prompt-LTL, une restriction de CLTL introduite dans [KPV09].

Ensuite, on montre l'équivalence entre WCMSO et CMSO, les versions quantitatives de Weak MSO et MSO, sur mots infinis, en généralisant la preuve de [KV01], et en définissant une forme normale pour les automates de coût sur mots infinis. Il est à noter que la preuve classique via le théorème de McNaughton n'est pas généralisable aux fonctions de coût, à cause de l'absence de modèle d'automate de coût déterministe. Pour montrer l'équivalence entre CMSO et les modèles d'automates de coût sur mots infinis, on généralise la complémentation de Büchi en décrivant une traduction des automates B-Büchi non-déterministes vers S-Büchi non-déterministes, et vice-versa.

On étudie finalement les fonctions de coût sur abres infinis. On introduit des automates de coût possédant les deux types de compteurs (B et S). On construit un transducteur permettant d'obtenir dans tous les cas une structure

spécifique sur ces automates : une hiérarchisation globale de leurs compteurs. Ceci nous permet d'obtenir le résultat principal de cette partie : une caractérisation de la classe des fonctions reconnues simultanément par automates B-Büchi et S-Büchi non-déterministes, au moyen des B-automates quasi-faibles, dont la définition constitue également une contribution. Ce résultat généralise celui de Rabin, développé par Kupferman et Vardi [Rab70, KV99], mais il est en partie inattendu. En effet, ce n'est pas la généralisation canonique (Weak MSO) qui s'applique, mais une généralisation hybride (les fonctions quasi-faibles), faisant intervenir explicitement le formalisme des fonctions de coût. On explicite finalement un corollaire de ce résultat, en revenant à la théorie des langages : étant donné un langage reconnu par automate de Büchi non-déterministe, on peut décider si son complémentaire peut également être reconnu par automate de Büchi non-déterministe (et donc par le théorème de Rabin, si ce langage est faible).

Plan de la thèse

On commence par présenter dans le Chapitre 1 la notion de fonction de coût de manière abstraite, indépendamment de tout formalisme de reconnaissance. On manipule cette notion en explicitant le lien avec les langages et en donnant des exemples.

La thèse se divise ensuite en trois parties, chacune correspondant au type des structures étudiées, sur un alphabet fini $\mathbb{A}$: mots finis, mots infinis, et enfin arbres infinis.

La Partie I concerne donc les mots finis, et commence par la description de la reconnaissance de fonctions de coût par automates. Ensuite, on développe plus longuement la notion de monoïde de stabilisation. Jusqu'ici, les résultats décrits sont ceux de Colcombet, et permettent de définir le cadre de la théorie des fonctions de coût régulières. Les contributions de la thèse commencent dans la Section suivante (2.3), où l'on définit les $\sharp$-expressions et $\omega\sharp$-expressions, et où l'on généralise aux fonctions de coût les notions de monoïde minimal et de congruence syntactique. On introduit enfin un troisième formalisme de reconnaissance des fonctions de coûts : les logiques de coût. Tous ces formalismes sont équivalents et définissent la classe des fonctions de coût régulières.

Le Chapitre 3 est consacrée à l'étude de la classe des fonctions temporelles, et le Chapitre 4 à celle des fonctions apériodiques. On termine la Partie I en étudiant l'intersection entre ces deux classes : on montre qu'elle est caractérisée par la logique Prompt-LTL.

On passe ensuite dans la Partie II à l'étude des mots infinis, toujours sur un alphabet fini. On décrit dans le Chapitre 6 les automates de coût dans ce nouveau contexte, en introduisant la condition de Büchi, ainsi que les automates alternants faibles. On définit les généralisations des logiques de coût CFO et CMSO aux mots infinis, et on introduit la logique faible WCMSO. Toutes les notions ci-dessus généralisent de manière canonique les notions classiques correspondantes. Le Chapitre 7 est consacré à la preuve de l'équivalence entre les deux formes duales (B et S) d'automates de Büchi non-déterministes. Ce

résultat était connu de Colcombet, sa rédaction détaillée apparaît ici par souci de complétude. Le Chapitre 8 généralise aux fonctions de coût le résultat classique selon lequel la logique faible et les automates alternants faibles reconnaissent tous les langages réguliers : on montre que toute fonction de coût régulière peut être reconnue par un B-automate faible alternant, et de manière équivalente par une formule de WCMSO. Enfin, le Chapitre 9 généralise le théorème de Kamp aux fonctions de coût, en montrant l'équivalence entre CLTL et CFO en termes de pouvoir expressif. On décrit aussi un modèle d'automates caractérisant cette classe de fonctions de coût.

On passe finalement à l'étude des fonctions de coût sur arbres infinis, qui constitue la Partie III. Notons que l'équivalence entre B- et S-automates n'a pas (encore ?) été obtenue sur les arbres infinis, on évitera donc de parler de fonction de coût régulière dans ce contexte. On commence le Chapitre 11 par donner un exemple de fonction de coût sur arbres infinis, avant de décrire les différents modèles d'automates qu'on utilisera. Ces automates généralisent ceux utilisés sur mots infinis, avec l'addition des B-automates alternants quasi-faibles, intermédiaires entre les automates faibles et les automates de Büchi. On donne quelques résultats sur ces automates à la fin du chapitre, en particulier on montre que la classe des fonctions quasi-faibles contient strictement celle des fonctions faibles. Dans le Chapitre 12, on introduit et on étudie des automates de coût possédant les deux types de compteurs (B et S). Le Chapitre 13 utilise ce résultat pour obtenir la caractérisation de la classe des fonctions quasi-faibles décrite plus haut. On explicite finalement dans la Section 13.3 un corollaire de ce résultat, selon lequel il existe un algorithme pour décider si le langage calculé par un automate de Büchi donné en entrée est faible.

Le Chapitre 14 conclut le document et ouvre des perspectives de continuations possibles de ce travail.

Notations

On note $\mathbb{N}$ l'ensemble des entiers positifs, et $\mathbb{N}_\infty := \mathbb{N} \cup \{\infty\}$, ordonné par $0 < 1 < \cdots < \infty$. Si n, m sont des entiers, l'intervalle d'entiers $\{n, n+1, \ldots, m\}$ sera noté $[n, m]$.

Dans la suite, $\mathbb{A}$ désignera un alphabet fini fixé, qui pourra être précisé dans des exemples. L'ensemble des mots finis sur $\mathbb{A}$ est noté $\mathbb{A}^*$, et le mot vide est noté ε. Par exemple si $\mathbb{A} = \{a, b\}$, on peut former les mots $aaaba$, ou $babbbaab$, qui sont des éléments de $\mathbb{A}^*$.

On note $\mathbb{A}^+ := \mathbb{A}^* \setminus \{\varepsilon\}$ l'ensemble des mots finis non vides. L'ensemble des mots infinis sur $\mathbb{A}$ est noté $\mathbb{A}^\omega$, par exemple $aababbaaaaa\ldots$ est un élément de $\mathbb{A}^\omega$, et sera noté $aababb(a)^\omega$. La concaténation des mots u et v est notée uv (si u est fini). L'exponentiation des mots se réfère à la concaténation, par exemple $aa(bab)^3b$ désignera le mot $aababbabbabb$, et pour tout mot $u \in \mathbb{A}^*$, $u^0 = \varepsilon$. Si u est un mot de $\mathbb{A}^*$, on utilisera aussi la notation u^* pour désigner l'ensemble de mots $\{u^n \ : \ n \in \mathbb{N}\} \subseteq \mathbb{A}^*$.

La longueur de u est notée $|u|$. Le nombre d'occurrences de la lettre a dans u est notée $|u|_a$. On aura ainsi $|aaba| = 4$, $|aaba|_b = 1$, $|\varepsilon| = 0$, et $|aab(ab)^\omega|_a = \infty$.

On utilisera aussi des arbres étiquetés par $\mathbb{A}$ dans la dernière partie de la thèse, les notations correspondantes seront définies seulement dans cette partie.

Les opérations de bornes inférieures et supérieures, notées inf et sup, seront effectuées sur l'ensemble $\mathbb{N}_\infty$, et ainsi on aura $\inf \emptyset = \infty$ et $\sup \emptyset = 0$.

Chapitre 1

Fonctions de coût

On commence par définir la notion de fonction de coût d'un point de vue mathématique, sans donner de procédé calculatoire. Ces fonctions sont introduites dans [Col09], et permettent de décrire des propriétés quantitatives de structures, en s'intéressant aux propriétés de bornes plutôt qu'aux valeurs exactes.

1.1 Relation d'équivalence

Soit E un ensemble, et $\mathcal{F}$ l'ensemble des fonctions : $E \to \mathbb{N}_\infty$. Une *fonction de correction* est une fonction croissante $\alpha : \mathbb{N}_\infty \to \mathbb{N}_\infty$, telle que $\alpha(\mathbb{N}) \subseteq \mathbb{N}$ et $\alpha(\infty) = \infty$. Les fonctions de correction seront notées $\alpha, \beta, \ldots$. Pour toute fonction de correction α, on définit une relation $\leq_\alpha$ sur $\mathbb{N}_\infty$ par $n \leq_\alpha m$ si $n \leq \alpha(m)$. Cette relation est naturellement étendue aux fonctions de $\mathcal{F}$: $f \preccurlyeq_\alpha g$ si $f \preccurlyeq \alpha \circ g$. Notons que $\preccurlyeq_\alpha$ n'est pas transitive : si $f \preccurlyeq_\alpha g$ et $g \preccurlyeq_\alpha h$, alors on peut seulement conclure que $f \preccurlyeq_{\alpha\circ\alpha} h$. On définit le préordre $\preccurlyeq$ sur $\mathcal{F}$ par $f \preccurlyeq g$ s'il existe α tel que $f \preccurlyeq_\alpha g$. Soit $\approx$ la relation d'équivalence induite par $\preccurlyeq$: $f \approx g$ si $f \preccurlyeq g$ et $g \preccurlyeq f$. On utilisera parfois la relation $\approx_\alpha$, définie par $f \approx_\alpha g$ si $f \preccurlyeq_\alpha g$ et $g \preccurlyeq_\alpha f$, pour préciser la fonction de correction utilisée. On note que $f \approx g$ si et seulement s'il existe α tel que $f \approx_\alpha g$.

Fait 1.1.1 *$f \approx g$ si et seulement si pour toute partie X de E, f est bornée sur X si et seulement si g est bornée sur X.*

Démonstration On note provisoirement $f \equiv g$ si pour toute partie X de E, f est bornée sur X si et seulement si g est bornée sur X. On veut donc montrer que les relations $\approx$ et $\equiv$ sont identiques.

Soit f et g deux fonctions de coût sur E. On supposue que $f \approx g$, c'est-à-dire qu'il existe α tel que $f \approx_\alpha g$. Si f est bornée par M sur un ensemble $X \subseteq E$, alors g est bornée par $\alpha(M)$ sur X (on utilise ici $\alpha(\mathbb{N}) \subseteq \mathbb{N}$). Par symétrie, pour tout $X \subseteq E$, si $g(X)$ est borné alors $f(X)$ est borné. On en conclut $f \equiv g$.

Réciproquement, supposons $f \equiv g$. On cherche à construire une fonction de correction α telle que $f \preccurlyeq_\alpha g$. Pour tout $n \in \mathbb{N}$, on pose

$$X_n = \{x : x \in E, g(x) \leq n\}, \text{ et } \alpha(n) := \sup f(X_n),$$

avec de plus $\alpha(\infty) = \infty$. On doit d'abord montrer que α est bien une fonction de correction : pour tout $n \in \mathbb{N}$, $\alpha(n) < \infty$, et α croissante. Soit $n \in \mathbb{N}$. Puisque $g(X_n)$ est borné par n et $f \approx g$, $f(X_n)$ est borné. Mais $\alpha(n)$ est défini comme $\sup f(X_n)$, donc $\alpha(n) < \infty$. De plus, on a bien α croissante : si $n \leq m$ on a $X_n \subseteq X_m$ donc $f(X_n) \subseteq f(X_m)$, et par conséquent $\alpha(n) = \sup f(X_n) \leq \sup f(X_m) = \alpha(m)$.

Soit $x \in E$ et $n = g(x)$, par définition de α on a $f(x) \leq \alpha(n) = \alpha(g(x))$. C'est vrai pour tout $x \in E$ donc $f \preccurlyeq_\alpha g$. Par symétrie, on peut obtenir une fonction de correction β telle que $g \preccurlyeq_\beta f$, et finalement $f \approx_{\max(\alpha,\beta)} g$.

□

Ainsi la relation $\approx$ préserve l'existence de bornes, tout en « oubliant » les valeurs exactes des fonctions : pour toute fonction de correction α, $f \approx \alpha \circ f$.

On définit les *fonctions de coût* sur E comme les élements de $\mathcal{F}/\approx$, c'est-à-dire les classes d'équivalence induites par la relation $\approx$. En pratique on désignera souvent une fonction de coût par l'un de ses représentants dans $\mathcal{F}$.

1.2 Lien avec les langages

Dans la suite, E sera un ensemble de structures (mots, arbres) étiquetées par un alphabet fini $\mathbb{A}$, ce qui justifie l'appellation « langage » pour désigner une partie de E. Pour un langage $L \subseteq E$, on note χ_L la fonction définie par $\chi_L(u) = 0$ si $u \in L$, et $\chi_L(u) = \infty$ si $u \notin L$. Ceci permet d'associer une fonction de coût à chaque langage, de telle sorte que $\chi_L \approx \chi_{L'}$ si et seulement si $L = L'$. Ainsi la théorie des fonctions de coût étend celle des langages. Cette extension est stricte, puisque par exemple la fonction qui à chaque mot associe sa longueur n'est équivalente à aucune fonction de la forme χ_L, pour $L \subseteq E$.

1.3 Exemples

On choisit pour E l'ensemble $\mathbb{A}^*$ des mots sur $\mathbb{A} := \{a, b\}$. On donne des exemples de fonctions de coût dénombrant diverses propriétés des mots sur $\mathbb{A}$. Les propriétés quantitatives les plus naturelles pour de tels mots sont par exemple leur longueur, ou le nombre d'occurence de « a ». On peut aussi s'intéresser à la taille des blocs de « a » consécutifs. On va voir comment ces exemples se comportent au sein de la théorie des fonctions de coût, pour manipuler un peu la relation d'équivalence $\approx$.

Soient $l, p, p', f_{\max}, f_{\min}, \text{minblock}_a, \text{maxblock}_a, g$ les fonctions $\mathbb{A}^* \to \mathbb{N}_\infty$ définies par :

- $l(u) := |u|$,
- $p(u) := |u|_a * |u|_b$,

- $p'(u) := (|u|_a + 1) * (|u|_b + 1)$,
- $f_{\max}(u) := \max(|u|_a, |u|_b)$,
- $f_{\min}(u) := \min(|u|_a, |u|_b)$,
- $\text{minblock}_a(a^{n_1}ba^{n_2}b \dots ba^{n_k}) := \min(n_1, n_2, \dots, n_k)$,
- $\text{maxblock}_a(a^{n_1}ba^{n_2}b \dots ba^{n_k}) := \max(n_1, n_2, \dots, n_k)$,
- $g(a^{n_1}ba^{n_2}b \dots ba^{n_k}) := \max(k, n_1, n_2, \dots, n_k)$.

Proposition 1.3.1 *On peut montrer les assertions suivantes :*

1. $l \approx p' \approx f_{\max}$*, c'est à dire que* l, p' *et* $f_{\max}$ *définissent la même fonction de coût.*
2. $p' \not\approx p$.
3. $\text{minblock}_a, \text{maxblock}_a$ *et* g *définissent trois fonctions de coût différentes.*
4. $g \approx l$.

Démonstration

1. Pour tout $u \in \mathbb{A}^*$, on a $f_{\max}(u) \leq l(u) \leq p'(u) \leq (f_{\max}(u) + 1)^2$. Ceci implique $l(u) \approx_\alpha p'(u) \approx_\alpha f_{\max}(u)$ avec $\alpha(n) = (n+1)^2$.
2. Pour l'ensemble de mots $X = a^*$, on a $p(X) = \{0\}$ mais $p'(X)$ non borné. En conséquence, X sépare p et p' : c'est un témoin de $p' \not\approx p$.
3. On définit les ensembles de mots $X_1 = a^*ba$ et $X_2 = (ab)^*$. Alors X_1 sépare minblock_a de maxblock_a et g, tandis que X_2 sépare maxblock_a de g.
4. Pour tout $u \in \mathbb{A}^*$, $g(u) \leq l(u) \leq g(u)^2$, donc $g \approx_\alpha l$, avec $\alpha(n) = n^2$.

□

On constate que la relation $\approx$ n'est pas toujours intuitive, et qu'il faut parfois examiner de près les fonctions que l'on définit pour avoir une idée de la fonction de coût (i.e. la classe d'équivalence) qu'elles représentent.

Première partie

Fonctions de coût régulières sur mots finis

Chapitre 2

Reconnaissance des fonctions régulières

On a généralisé la notion de langage pour permettre de décrire des propriétés quantitatives des structures étudiées. Dans la théorie des langages, la notion de régularité est centrale. Un langage est régulier s'il est reconnu par un automate fini, ou de manière équivalente un semigroupe fini, une expression régulière, une formule de la logique monadique de second ordre (MSO). La généralisation de cette notion aux fonctions de coût est effectuée dans [Col09].

Nous décrirons ici plusieurs formalismes équivalents permettant de reconnaitre les fonctions de coût régulières : les automates de coût (possédant deux sémantiques duales), les semigroupes de stabilisation, et des versions quantitatives des logiques FO, MSO et LTL. On considèrera ces formalismes comme agissant parfois sur $\mathbb{A}^*$, parfois seulement sur $\mathbb{A}^+$, le mot vide ε pouvant amener des complications d'ordre technique sans réelle importance.

On définira également les $\omega\sharp$-expressions, et on montrera comment obtenir le semigroupe de stabilisation minimal d'une fonction de coût régulière au moyen d'une relation de congruence sur ces $\omega\sharp$-expressions.

2.1 Automates de coût

Les premiers modèles d'automates à poids ont été initialement définis dans [Sch61]. Dans de tels automates, chaque arête est pondéré par un certain poids, représenté par un nombre entier ou réel. On peut ainsi associer un poids à un chemin dans l'automate, en sommant tous les poids des arêtes. De plus, ces automates sont non-déterministes : il peut exister plusieurs manières de lire un mot, et donc des choix à effectuer. Pour des automates classiques, on donne un sens au non-déterminisme en définissant l'accpetation d'un mot comme l'existence d'un chemin acceptant. Ici, le but de l'automate non-déterministe est non seulement de trouver un chemin acceptant, mais également de poids optimal (ceci signifiant traditionnellement de poids minimum).

Les automates de coût, introduits pour la première fois dans [BC06], constituent une généralisation des automates à poids. Leur nouveauté est d'autoriser les transitions à remettre à zéro le poids courant, et également de donner à l'automate la possibilité d'avoir plusieurs compteurs distincts, ayant chacun un poids associé à la fin de l'exécution. Cependant, puisque les valeurs exactes seront ignorées par la relation d'équivalence $\approx$, les poids 0 et 1 suffisent pour étiqueter les transitions. On va définir deux formes duales d'automates de coût, reflétant des conceptions différentes pour les notions de poids le long d'un chemin et de chemin optimal.

2.1.1 Compteurs

Les automates de coût généralisent les automates classiques par l'ajout d'un ensemble fini de compteurs, dont les valeurs sont actualisées à chaque transition. Les compteurs seront notés $\gamma, \gamma_1, \ldots$.

Au début de l'exécution, chaque compteur contient la valeur 0. Les actions atomiques sur les compteurs sont

- **Incrément** noté `i`, augmente d'une unité la valeur du compteur.
- **Reset** noté `r`, remet la valeur du compteur à 0.
- **Check** noté `c`, enregistre la valeur du compteur sans la changer (utilisé pour la sémantique de l'automate).

On conserve ici la terminologie anglo-saxonne « Reset » et « Check », par souci de cohérence avec le reste de la littérature.

Une *action* est un mot sur l'alphabet $\{\mathtt{i}, \mathtt{r}, \mathtt{c}\}$, qui effectue chaque action atomique les unes à la suite des autres. Par exemple l'action `iicr` augmente de 2 la valeur du compteur, l'enregistre, puis la remet à zéro. En particulier l'action ε correspondant au mot vide laisse le compteur inchangé.

Une action sur un ensemble de k compteurs est un k-uplet d'actions, chacune s'appliquant à un compteur.

Si ν est une action sur un ensemble de compteurs, on note $C(\nu)$ l'ensemble des valeurs enregistrées (au moyen de l'action **check**) durant cette action (les valeurs étant initialisées à 0). Remarquons que l'ordre et la provenance des valeurs enregistrées est oubliée, ainsi que les répétitions éventuelles.

On utilisera en fait ces compteurs sous deux formes plus restreintes : les B-compteurs et les S-compteurs.

B-compteurs :

Un B-compteur est un compteur restreint aux actions atomiques $\mathbb{C}_B := \{\varepsilon, \mathtt{ic}, \mathtt{r}\}$. Une action globale effectuée sur un ensemble Γ de B-compteurs est un mot $(\mathbb{C}_B^\Gamma)^*$. On définit la valeur d'une action globale $\nu \in (\mathbb{C}_B^\Gamma)^*$ par

$$val_B(\nu) = \sup C(\nu).$$

Cela signifie que la valeur d'une telle action est la plus grande valeur enregistrée au cours de l'action, et 0 si aucune valeur n'a été enregistrée.

Par exemple $val_B(\mathtt{ic.ic.r}.\varepsilon.\mathtt{ic.r.ic}) = 2$, et $val_B(\varepsilon.\varepsilon.\mathtt{r}.\varepsilon) = 0$.

S-compteurs :

Un S-compteur est un compteur restreint aux actions atomiques $\mathbb{C}_S := \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}$. Une action globale effectuée sur un ensemble Γ de S-compteurs est un mot $(\mathbb{C}_S^\Gamma)^*$. On définit la valeur d'une action globale $\nu \in (\mathbb{C}_S^\Gamma)^*$ par

$$val_S(\nu) = \inf C(\nu).$$

Cela signifie que la valeur d'une telle action est la plus petite valeur enregistrée au cours de l'action, et ∞ si aucune valeur n'a été enregistrée. Par exemple $val_S(\mathtt{i.i.i.r.i.i.cr.i.r.i.}\varepsilon\mathtt{.i.i.cr}) = 2$, et $val_S(\varepsilon\mathtt{.i.i.r}) = \infty$.

En utilisant ces deux types de compteurs, on définit deux formes duales d'automates.

2.1.2 B-automates

Un B-automate est un tuple $\langle Q, \mathbb{A}, \mathit{In}, \mathit{Fin}, \Gamma, \Delta\rangle$, où Q est un ensemble fini d'états, $\mathbb{A}$ est l'alphabet fini fixé, *In* et *Fin* sont les ensembles d'états initiaux et finaux, Γ est un ensemble fini de compteurs, et $\Delta \subseteq Q \times \mathbb{A} \times \mathbb{C}_B^\Gamma \times Q$ est l'ensemble des transitions.

Une *exécution* R d'un tel automate sur un mot $u = a_1 a_2 \dots a_n \in \mathbb{A}^*$ est une séquence $p_0, \nu_1, p_1, \nu_2, p_2, \dots, \nu_n, p_n$ telle que $p_0 \in \mathit{In}$, $p_n \in \mathit{Fin}$, et pout tout $i \in [1, n]$, $(p_{i-1}, a_i, \nu_i, p_i) \in \Delta$.

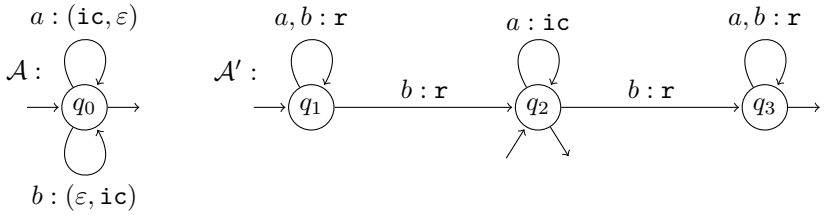
On étend ensuite la définition de val_B aux exécutions :
si $R = p_0, \nu_1, p_1, \nu_2, p_2, \dots, \nu_n, p_n$ est une exécution de $\mathcal{A}$ alors $val_B(R) := \sup C(\nu_1\nu_2\dots\nu_n)$. Autrement dit la valeur d'une exécution est la plus haute valeur enregistrée au cours de cette exécution, tous compteurs confondus.

Finalement, on définit la sémantique d'un B-automate $\mathcal{A}$ de la manière suivante : la fonction $\mathbb{A}^* \to \mathbb{N}_\infty$ calculée par $\mathcal{A}$ est notée $[\![\mathcal{A}]\!]_B$, et est donnée par

$$[\![\mathcal{A}]\!]_B(u) := \inf \{val_B(R) : R \text{ exécution de } \mathcal{A} \text{ sur } u\}$$

Les fonctions calculées par des B-automates seront toujours considérées comme des fonctions de coût, c'est-à-dire modulo $\approx$. On dira qu'un B-automate $\mathcal{A}$ *reconnaît* une fonction de coût f si $[\![\mathcal{A}]\!]_B \approx f$.

Exemple 2.1.1 *Soit* $\mathbb{A} = \{a, b\}$. *Les automates* $\mathcal{A} := \langle \{q_0\}, \mathbb{A}, \{q_0\}, \{q_0\}, \{\gamma_1, \gamma_2\}, \Delta\rangle$ *et* $\mathcal{A}' := \langle \{q_1, q_2, q_3\}, \mathbb{A}, \{q_1, q_2\}, \{q_2, q_3\}, \{\gamma\}, \Delta'\rangle$ *suivants reconnaissent respectivement* $f_{\max}$ *et* minblock_a *définies dans la Section 1.3 :*



L'automate $\mathcal{A}$ est déterministe, et possède deux compteurs γ_1 et γ_2. Pour tout mot u, sa seule exécution stockera le nombre de a dans γ_1, et le nombre de b dans γ_2. Puisque la B-sémantique effectue un maximum sur tous les compteurs, la fonction calculée est bien $[\![\mathcal{A}]\!]_B(u) = \max(|u|_a, |u|_b)$.

En revanche, l'automate $\mathcal{A}'$ est non-déterministe, et ne possède qu'un seul compteur. Le seul choix non-déterministe est le moment où l'automate passe dans l'état q_2. L'automate compte ensuite le nombre de a consécutifs, jusqu'au prochain b qui le fait passer en q_3. Chaque exécution mesure donc la taille d'un bloc de a. Puisque la sémantique $[\![\mathcal{A}]\!]_B$ est définie comme un infimum sur toutes les exécutions, la fonction calculée par $\mathcal{A}'$ est bien minblock_a.

Remarque 2.1.2 *Contrairement au cas classique des langages, le non-déterminisme est ici nécessaire pour capturer toute la classe des fonctions calculées par B-automates. En effet, c'est le seul moyen de calculer le minimum de plusieurs quantités, comme le fait l'automate $\mathcal{A}'$ de l'exemple 2.1.1. On peut montrer qu'aucun B-automate déterministe ne reconnaît la fonction de coût* minblock_a.

2.1.3 S-automates

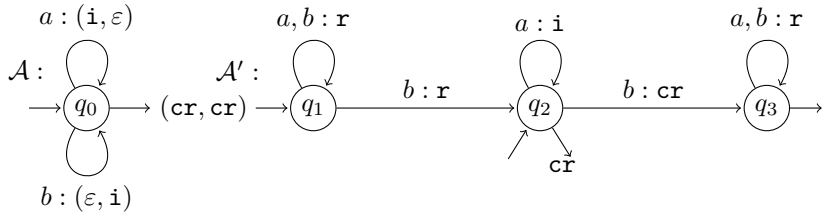
Il est difficile de généraliser la complémentation aux fonctions de coût, car cette notion n'a plus vraiment de sens. Cependant, on peut définir une sémantique duale de celles des B-automates, qui correspond à la complémentation si on se restreint aux langages. Le principe est d'inverser les rôles de inf et sup.

On définit les S-automates de la même manière que les B-automates, avec l'ensemble d'actions atomiques $\mathbb{C}_S := \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}$. De plus, on autorise une action supplémentaire effectuée à la fin du mot, marquée sur la flèche de sortie de l'état final. Cette action finale ne change pas l'expressivité du modèle, mais elle permet de définir de manière plus naturelle certains automates (voir l'exemple suivant).

La sémantique d'un S-automate $\mathcal{A}$ est la fonction de coût $[\![\mathcal{A}]\!]_S$ définie par

$$[\![\mathcal{A}]\!]_S(u) := \sup\{val_S(R) : R \text{ exécution de } \mathcal{A} \text{ sur } u\}.$$

Exemple 2.1.3 *Soit $\mathbb{A} = \{a, b\}$. Les automates $\mathcal{A} := \langle \{q_0\}, \mathbb{A}, \{q_0\}, \{q_0\}, \{\gamma_1, \gamma_2\}, \Delta\rangle$ et $\mathcal{A}' := \langle \{q_1\}, \mathbb{A}, \{q_1\}, \{q_1\}, \{\gamma\}, \Delta'\rangle$ suivants reconnaissent respectivement $f_{\min}$ et* maxblock_a *:*



Ces automates comptent les mêmes évènements que ceux de l'Exemple 2.1.1, mais la S-sémantique inverse les opérations de maximum et de minimum par

rapport à la B-sémantique. Il est donc facile de voir que les fonctions définies ici sont $f_{\min}$ et $\mathrm{maxblock}_a$.

Remarque 2.1.4 *Si $L \subseteq \mathbb{A}^*$ est un langage reconnu par un automate $\mathcal{A}$, alors $\mathcal{A}$ peut être vu comme un B-automate sans compteur avec $[\![\mathcal{A}]\!]_B = \chi_L$, et également comme un S-automate sans compteur avec $[\![\mathcal{A}]\!]_S = \chi_{\overline{L}}$.*

Théorème 2.1.5 ([BC06, Col09]) *La classe des fonctions de coût reconnues par B-automates est égale à celle des fonctions de coût reconnues par S-automates. Les fonctions de coût de cette classe seront dites* régulières.

La classe des fonctions de coût régulières est une généralisation de celle des langages réguliers. Le lemme suivant décrit plus précisément la relation entre ces deux notions :

Lemme 2.1.6 *Soit $L \subseteq \mathbb{A}^*$ un langage, alors les deux assertions suivantes sont équivalente :*

- *L est un langage régulier,*
- *χ_L est une fonction de coût régulière.*

Démonstration Si L est régulier, alors il est reconnu par un automate fini $\mathcal{A}$. On a vu que si l'on considère $\mathcal{A}$ comme un B-automate, alors $[\![\mathcal{A}]\!]_B = \chi_L$, ce qui montre que χ_L est une fonction de coût régulière. Réciproquement, on suppose que χ_L est une fonction de coût régulière, et donc qu'il existe un B-automate $\mathcal{A} = \langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta\rangle$ tel que $[\![\mathcal{A}]\!]_B \approx \chi_L$. Cela signifie qu'il existe $n \in \mathbb{N}$ tel que pour tout $u \in L$, $[\![\mathcal{A}]\!]_B(u) \leq n$, et pour tout $u \notin L$, $[\![\mathcal{A}]\!]_B(u) = \infty$. On va décrire comment construire un automate fini $\mathcal{A}'$ pour L. Cet automate est constitué de $(n+1)^{|\Gamma|}$ copies de $\mathcal{A}$, et comporte une copie notée $\mathcal{A}_v$ pour chaque $v \in [0,n]^\Gamma$. Le principe est de stocker dans les états la valeur de chacun des compteurs de $\mathcal{A}$, jusqu'à la valeur n. Ainsi, les états initiaux seront ceux de $\mathcal{A}_{(0,\ldots,0)}$, et à tout moment l'indice de la copie courante contiendra la valuation des compteurs. Si l'un des compteurs dépasse n, il n'y a pas de transition disponible, et l'exécution échoue. Ainsi, les exécutions acceptantes de $\mathcal{A}'$ sont en bijection avec les exécutions de $\mathcal{A}$ de valeur au plus n. Puisque les mots pour lesquels il existe une telle exécution de $\mathcal{A}$ sont exactement ceux de L, le langage reconnu par $\mathcal{A}'$ est bien L. On peut en conclure que L est régulier.

□

2.1.4 Opérations sur les fonctions de coût régulières

Etant données deux fonctions de coût f et g reconnues par des automates $\mathcal{A}$ et $\mathcal{B}$, il est naturel de vouloir construire un automate reconnaissant par exemple $\min(f,g)$, $\max(f,g)$, et pourquoi pas d'autres combinaisons de f et g.

De même que dans le cas classique, on peut définir l'union et le produit de B-automates ou S-automates. L'union est obtenu simplement en accolant deux automates, avec un choix non-déterministe au début pour décider lequel des

deux sera utilisé. Le produit est obtenu en faisant fonctionner simultanément les deux automates, l'ensemble des états Q sera donc l'ensemble produit $Q_{\mathcal{A}} \times Q_{\mathcal{B}}$, et l'ensemble des compteurs sera $\Gamma = \Gamma_{\mathcal{A}} \cup \Gamma_{\mathcal{B}}$.

Par la définition des sémantiques des B- et S-automates, il est facile d'obtenir les propriétés suivantes :

Proposition 2.1.7 ([Col09]) *Si $\mathcal{A}$ et $\mathcal{B}$ sont deux B-automates, alors*
- *leur union reconnait* $\min([\![\mathcal{A}]\!], [\![\mathcal{B}]\!])$*, car le non-déterminisme se résoud en un minimum.*
- *leur produit reconnait* $\max([\![\mathcal{A}]\!], [\![\mathcal{B}]\!])$*, car seul le compteur de plus grande valeur est considéré.*

De manière duale, si $\mathcal{A}$ et $\mathcal{B}$ sont deux S-automates, alors
- *leur union reconnait* $\max([\![\mathcal{A}]\!], [\![\mathcal{B}]\!])$
- *leur produit reconnait* $\min([\![\mathcal{A}]\!], [\![\mathcal{B}]\!])$.

On va s'intéresser à une troisième opération sur les fonctions de coût : la projection. L'idée intuitive est de vouloir « oublier » une partie de l'information contenue dans le mot.

Soit $f : \mathbb{A}^* \to \mathbb{N}_\infty$, $\mathbb{B}$ un alphabet fini, et $h : \mathbb{A} \to \mathbb{B}$ une fonction étendue en morphisme $h : \mathbb{A}^* \to \mathbb{B}^*$ préservant la longueur des mots.

On définit l'inf-projection $f_{\inf,h} : \mathbb{B}^* \to \mathbb{N}_\infty$ (resp. sup-projection $f_{\sup,h} : \mathbb{B}^* \to \mathbb{N}_\infty$) de f suivant h par

$$f_{\inf,h}(v) = \inf \{f(u) : h(u) = v\} \text{ et } f_{\sup,h}(v) = \sup \{f(u) : h(u) = v\}.$$

Si $\mathcal{A}$ est un automate sur alphabet $\mathbb{B}$, on définit sa projection suivant h, notée $h^{-1}(\mathcal{A})$, comme l'automate $\mathcal{A}$ où chaque transition étiquetée b est remplacée par un ensemble de transitions étiquetées par les lettres a telles que $h(a) = b$. Intuitivement, l'automate $h^{-1}(\mathcal{A})$ a accès à une information plus détaillée (l'alphabet $\mathbb{A}$), mais se comporte comme un automate qui n'aurait accès qu'à l'image des lettres par h.

De même que précédemment, la définition des B- et S-sémantiques a pour conséquence la proposition suivante :

Proposition 2.1.8 ([Col09]) *Soit $\mathcal{A}$ un B-automate, $\mathcal{A}'$ un S-automate, et $h : \mathbb{A} \to \mathbb{B}$, alors*
- $h^{-1}(\mathcal{A})$ *reconnaît* $[\![\mathcal{A}]\!]_{\inf,h}$
- $h^{-1}(\mathcal{A}')$ *reconnaît* $[\![\mathcal{A}']\!]_{\sup,h}$

Toutes les propositions ci-dessus restent vraies si on considère les fonctions exactes, et non plus les fonctions de coût (modulo $\approx$). L'approximation $\approx$ est nécessaire durant le passage de B à S (et vice-versa), mais n'intervient pas dans ces constructions.

Théorème 2.1.9 ([Col09]) *Si f et g sont deux fonctions de coût régulières, le problème de savoir si « $f \preccurlyeq g$ » est décidable.*

En particulier, les problèmes « $f \approx g$ », et « f bornée » sont décidables.

Remarquons que la décidabilité du problème « f bornée ? » s'obtient par le fait que f est bornée si et seulement si $f \preccurlyeq 0$.

Le principe qui sous-tend ce théorème est que la question « $f \preccurlyeq g$? » devient plus simple si f est donnée par un S-automate $\mathcal{A}$, et g par un B-automate $\mathcal{B}$. En effet, elle revient à la recherche d'un contre-exemple, c'est-à-dire d'une famille de mots de valeurs arbitrairement grande pour $\mathcal{A}$ et bornée pour $\mathcal{B}$. Par la sémantique des B- et S-automates, cela se réduit à chercher un chemin vérifiant certaines propriétés dans $\mathcal{A} \times \mathcal{B}$, ce qui permet de donner un algorithme pour répondre à la question posée.

2.2 Semigroupes de stabilisation

2.2.1 Rappels classiques

Un *semigroupe ordonné* $\mathbf{S} = \langle S, \cdot, \leq \rangle$ est un ensemble S muni d'un produit associatif $\cdot : S \times S \to S$ et d'un ordre partiel $\leq$ compatible avec le produit (i.e. si $a \leq a'$ et $b \leq b'$, alors $a \cdot b \leq a' \cdot b'$). Si $\langle S, \cdot \rangle$ contient un élément neutre, on dit que $\mathbf{S}$ est un *monoïde* et le neutre est noté 1. On note $\mathbf{S}^1$ le monoïde obtenu par l'ajout de 1 à $\mathbf{S}$ si nécessaire (sinon $\mathbf{S}^1 = \mathbf{S}$).

Un *idempotent* de $\mathbf{S}$ est un élément $e \in S$ tel que $e \cdot e = e$. On note $E(\mathbf{S})$ l'ensemble des idempotents de $\mathbf{S}$.

Reconnaissance de langages

Dans la théorie standard, la reconnaissance d'un langage par un semigroupe fini se fait un moyen d'un morphisme de l'ensemble des mots vers un semigroupe fini. On peut décomposer ce processus en deux étapes. Premièrement,un morphisme $h : \mathbb{A}^+ \to S^+$ préservant la longueur, transforme les mots sur $\mathbb{A}$ en mots sur S. Ensuite, on utilise le produit généralisé $\pi : S^+ \to S$ qui associe à chaque mot sur S le produit de ses lettres (étant associatif, le produit sur S^+ n'est pas ambigü). Le langage L reconnu par le triplet (S, h, P), où P est une partie de S, est $L := h^{-1}(\pi^{-1}(P))$, c'est-à-dire $u \in L$ si et seulement si $\pi(h(u)) \in P$.

La classe des langages reconnus par des semigroupes finis est exactement celle des langages réguliers. Certaines classes de langages peuvent être caractérisées par des restrictions sur les semigroupes qui les reconnaissent, comme précisé par le théorème d'Eilenberg [Eil74]. Un exemple plus précis est celui du théorème de Schützenberger [Sch65], qui établit la correspondance entre les langages définissables par une expression rationnelle sans étoile, et ceux reconnus par un semigroupe apériodique.

On voudrait donc généraliser les semigroupes aux fonctions de coût, pour disposer d'un outil puissant permettant de caractériser et décider de nombreuses propriétés, comme c'est le cas pour les langages.

2.2.2 Semigroupes de stabilisation

La notion de semigroupe de stabilisation a été introduite dans [Col09] comme extension quantitative des semigroupes standards, dans le but de reconnaître des fonctions de coût. Cette notion est fortement inspirée de travaux précédents [Sim78, Has90, LP04, Kir05] qui ont permis de concevoir et affiner progressivement une approche algébrique de propriétés quantitatives liées à l'existence de bornes. Elle a également été utilisée dans des contextes légèrement différents, comme celui des automates probabilistes [FGO12]. Dans tous les cas, l'idée sous-jacente est d'ajouter aux semigroupes une opération dite de stabilisation, qui permet de décrire des propriétés quantitatives de certains éléments. Cette opération permet de spécifier que même si un élément est idempotent, le répéter de nombreuses fois nous permet d'obtenir un autre élément, au comportement différent. Cette opération est donc a priori en désaccord avec le produit, puisque si e est idempotent, pour tout $n > 0$, $e^n = e$. On commence par définir formellement les semigroupes de stabilisation, puis on verra dans la suite comment les mots peuvent être évalués dans ce cadre, et enfin comment un semigroupe de stabilisation peut reconnaître une fonction de coût.

Définition 2.2.1 *Un* semigroupe de stabilisation $\langle S, \cdot, \leq, \sharp \rangle$ *est un semigroupe ordonné* $\langle S, \cdot, \leq \rangle$ *muni d'un opérateur unaire* $\sharp \colon E(\mathbf{S}) \to E(\mathbf{S})$ *(appelé* stabilisation*) vérifiant :*

- *pour tous* $a, b \in S$ *avec* $a \cdot b \in E(\mathbf{S})$ *et* $b \cdot a \in E(\mathbf{S})$*, on a* $(a \cdot b)^\sharp = a \cdot (b \cdot a)^\sharp \cdot b$ *;*
- *pour tout* $e \in E(\mathbf{S})$*,* $(e^\sharp)^\sharp = e^\sharp \leq e$ *;*
- *pour tous* $e \leq f$ *dans* $E(\mathbf{S})$*,* $e^\sharp \leq f^\sharp$*.*

Si de plus $\mathbf{S}$ *est un monoïde d'élément neutre* 1 *et* $1^\sharp = 1$*, on dit que* $\mathbf{S}$ *est un* monoïde de stabilisation*.*

On considèrera seulement des semigroupes de stabilisation finis. On peut se représenter l'opérateur $\sharp$ comme répétant « un grand nombre de fois » son argument. Cette intuition est reflétée dans les propriétés suivantes, conséquences de la définition :

$$e^\sharp = e \cdot e^\sharp = e^\sharp \cdot e = e^\sharp \cdot e^\sharp = (e^\sharp)^\sharp \leq e$$

La relation d'ordre $a \leq b$ signifie intuitivement que la seule distinction entre a et b provient de considérations quantitatives : certains évènements se produisent en plus grand nombre dans a que dans b.

Puisque l'on s'intéresse maintenant aux propriétés quantitatives des mots, on a besoin d'utiliser un outil plus subtil que le produit π utilisé dans le cas classique, pour passer d'un mot de S^+ à un élément de S. Si on se contentait d'effectuer le produit, un idempotent e répété un grand nombre de fois ne serait pas distinguable d'une unique occurence de e. Le rôle de la stabilisation est de distinguer « peu » de « beaucoup », mais étant donné un mot de S^+, il reste encore à définir formellement à quel moment on appliquera l'opérateur $\sharp$.

Structure

Soit $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ un semigroupe de stabilisation.

Pour analyser finement la structure algébrique définie par les opérations de composition et de stabilisation, nous définissons les relations de Green [Gre51] développées dans le cadre de la théorie des monoïdes.

Pour $a, b \in S$, on note $a \leq_{\mathcal{J}} b$ s'il existe $x, y \in \mathbf{S}^1$ tels que $a = xby$. La relation $\leq_{\mathcal{J}}$ est un préordre. On définit la relation d'équivalence $\mathcal{J}$ comme $\leq_{\mathcal{J}} \cap \geq_{\mathcal{J}}$. La relation $\leq_{\mathcal{J}}$ induit un ordre sur les $\mathcal{J}$-classes, également noté $\leq_{\mathcal{J}}$.

Un *élément régulier* de $\mathbf{S}$ est un élément $\mathcal{J}$-équivalent à un idempotent. On dira donc qu'une $\mathcal{J}$-classe qui contient un idempotent est régulière (tous ses éléments sont réguliers). Dans le cas contraire, on dira que la $\mathcal{J}$-classe est *irrégulière*.

On peut étendre $\sharp$ à tous les éléments réguliers de S. Si a est régulier, il existe $e \in E(\mathbf{S})$ et $x, y \in S \cup \{1\}$ tels que $x \cdot e \cdot y = a$. On définit alors $a^\sharp = x \cdot e^\sharp \cdot y$. Le résultat ne dépend pas du choix de e, x, y (*cf.* [Kir05]).

Définition 2.2.2 *Une $\mathcal{J}$-classe régulière J est* stable *s'il existe un idempotent a dans J tel que $a^\sharp \in J$. Dans le cas contraire elle est* instable.

Lemme 2.2.3 *Si J est stable, alors pour tout $a \in J$, $a^\sharp = a$.*

On dira également qu'un élément régulier a est stable si $a^\sharp = a$, et instable sinon. Remarquons que si a est un idempotent instable, alors $a^\sharp \not\mathcal{J} a$, et $a^\sharp \cdot a = a^\sharp$, donc $a^\sharp <_{\mathcal{J}} a$.

2.2.3 Arbres de calcul

Le théorème des forêts de factorisation , démontré par Simon [Sim90], permet de décrire l'évaluation structurée de mots dans le cadre de semigroupes classiques. Cette évaluation structurée est décrite par des arbres de hauteur bornée, dont les seuls noeuds non binaires doivent avoir des fils tous identiques et idempotents. Ce théorème peut être vu comme une propriété de type Ramsey sur les semigroupes finis : il existe toujours un moyen de regrouper les éléments selon une structure précise, dans tout semigroupe fini. Cette propriété des semigroupes finies a pu être utilisée pour d'autres travaux, voir par exemple [PW95, Sim94, BC06]. Colcombet a généralisé ce théorème sur les arbres [Col07a] et sur les mots infinis [Col07b], ainsi que dans le cadre des semigroupes de stabilisation [Col11]. Nous présentons ici cette dernière contribution, qui sera ensuite utilisée pour donner une sémantique aux semigroupes de stabilisation.

Soit $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ un semigroupe de stabilisation, et $u \in S^+$. Un *n-calcul* t sur u est un arbre étiqueté par S tel que le mot de feuilles de t est u, et pour chaque noeud p de t, l'un de ces cas est vérifié :

Feuille p est une feuille,

Binaire : p a exactement 2 fils p_1, p_2, et $t(p) = t(p_1) \cdot t(p_2)$,

Idempotent : p a k fils $p_1, \dots, p_k$ avec $k \leq n$, et il existe $e \in E(\mathbf{S})$ tel que $t(p) = t(p_1) = \cdots = t(p_k) = e$,

Stabilisation : p a k fils $p_1, \dots, p_k$ avec $k > n$, et il existe $e \in E(\mathbf{S})$ tel que $t(p_1) = \cdots = t(p_k) = e$, et $t(p) = e^\sharp$.

L'étiquette de la racine de t est appelée sa *valeur* et est notée $val(t)$. En d'autres termes, un arbre de calcul décrit la manière dont on évalue un mot dans un semigroupe de stabilisation. On peut multiplier deux lettres, ou un certain nombres d'idempotents. Faire un arbre de n-calcul revient à considérer qu'une concaténation de plus de n idempotents consécutifs est « grande », et l'on doit appliquer l'opérateur $\sharp$ pour refléter cette information.

On peut remarquer que si l'on ne borne pas la hauteur, on ne sera jamais forcé d'utilisé des noeuds de stabilisation. On va donc seulement s'intéresser aux n-calculs ne dépassant pas une certaine hauteur.

Théorème 2.2.4 ([Col11]) *Pour tout* $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$, $u \in S^+$ *et* $n \in \mathbb{N}$, *il existe un* n*-calcul sur* u *de hauteur au plus* $H = 3|S|$.

On va donc considérer que les seuls calculs valides sont ceux de hauteur bornée, pour l'instant par H. Dans la suite on omet de rappeler cette contrainte, mais l'appellation « n-calcul » signifie maintenant implicitement « n-calcul de hauteur au plus H ».

On veut voir la valeur d'un calcul (l'étiquette de sa racine) comme l'évaluation d'un mot de S^+, c'est-à-dire la généralisation du produit. Il y a cependant plusieurs problèmes pour l'instant :

- L'évaluation d'un mot dépend du n que l'on se fixe comme seuil, au moyen d'un n-calcul.
- La hauteur des calculs n'est pas flexible, or, on aimerait pouvoir les composer simplement.
- On veut refléter seulement des propriétés de borne, dans l'esprit de la relation $\approx$.

Les sections suivantes apportent des solutions à ces problèmes, en introduisant du formalisme permettant de manipuler finement des calculs. On définit d'abord les suites de coût, qui reflètent le comportement de la suite de valeurs de n-calculs sur un même mot u, pour n prenant toutes les valeurs entières. Ceci nous permet de donner une sémantique précise aux semigroupes de stabilisation, via les notions d'idéaux et de fonctions compatibles.

2.2.4 Suites de coût

On introduit la notion de suite de coût pour pouvoir étudier le comportement d'un ensemble de n-calculs, pour n variable. Historiquement, les suites de coût [Col09] sont apparues avant les arbres de calculs [Col11] dans la théorie des fonctions de coût, et avaient pour but de donner directement une sémantique aux semigroupes de stabilisation. On a choisi ici de renverser cet ordre, les arbres de calculs étant jugés plus intuitifs.

Soit $(E, \leq)$ un ensemble ordonné. Dans la pratique E sera un semigroupe ordonné, mais seulement l'ordre est nécessaire dans cette partie. Ceci nous permet de définir les suites de coût de manière plus générale que le contexte restreint des semigroupes.

Soit α une fonction de correction, et $\vec{a}, \vec{b} \in E^{\mathbb{N}}$ deux suites d'éléments de E. On définit la relation $\preceq_\alpha$ sur $E^{\mathbb{N}}$ par $\vec{a} \preceq_\alpha \vec{b}$ si :

$$\forall n. \forall m. \quad \alpha(n) \leq m \rightarrow \vec{a}(n) \leq \vec{b}(m) \ .$$

Une suite $\vec{a}$ est dite *α-croissante* si $\vec{a} \preceq_\alpha \vec{a}$. On définit la relation $\sim_\alpha := \preceq_\alpha \cap \succeq_\alpha$. On pourra omettre la fonction de correction : on notera $\vec{a} \preceq \vec{b}$ (resp. $\vec{a} \sim \vec{b}$) s'il existe α tel que $\vec{a} \preceq_\alpha \vec{b}$ (resp. $\vec{a} \sim_\alpha \vec{b}$).

Remarques 2.2.5
- *si $\alpha \leq \alpha'$ alors $\vec{a} \preceq_\alpha \vec{b}$ implique $\vec{a} \preceq_{\alpha'} \vec{b}$,*
- *Une suite est α-croissante si et seulement si elle est $\sim_\alpha$-équivalente à une suite croissante,*
- *si $\vec{a}, \vec{b} \in E^{\mathbb{N}}$ sont croissantes, alors $\vec{a} \preceq_\alpha \vec{b}$ si et seulement si $\vec{a} \circ \alpha \leq \vec{b}$.*

Les suites α-croissantes, ordonnées par $\preceq_\alpha$ peuvent être vues comme une généralisation du cas $\alpha = id$.

les relations $\preceq_\alpha$ et $\sim_\alpha$ ne sont pas transitives, mais $\preceq$ et $\sim$ le sont, comme le montre la propriété suivante :

Proposition 2.2.6
- *$\vec{a} \preceq_\alpha \vec{b} \preceq_\alpha \vec{c}$ implique $\vec{a} \preceq_{\alpha\circ\alpha} \vec{c}$*
- *$\vec{a} \sim_\alpha \vec{b} \sim_\alpha \vec{c}$ implique $\vec{a} \sim_{\alpha\circ\alpha} \vec{c}$.*

On peut voir la fonction de correction α comme un paramètre de précision pour $\sim$ et $\preceq$. La proposition 2.2.6 montre qu'un pas de transitivité résulte en une perte de précision.

On identifiera dans la suite les éléments $a \in E$ avec la suite constante de valeur a. Ceci nous permettra de considérer les suites α-croissantes comme une extension de E.

Pour tout α, la relation $\preceq_\alpha$ coïncide avec $\leq$ sur les suites constantes (modulo l'identification ci-dessus). En conséquence, $(E^{\mathbb{N}}, \preceq_\alpha)$ est une extension de $(E, \leq)$.

Définition 2.2.7 *Soit $a, b \in E$ avec $a \leq b$, et $n \in \mathbb{N}$, on définit la suite $a|_n b$ par :*

$$(a|_n b)(k) = \begin{cases} a & \text{si } k < n, \\ b & \text{sinon.} \end{cases}$$

2.2.5 Fonctions à valeurs dans les suites de coûts

Pour évaluer un mot u, on a vu qu'il reste à fixer un seuil $n \in \mathbb{N}$ afin de pouvoir construire un n-calcul sur u. Afin de caractériser toutes les évaluations possibles de u, on peut donc s'intéresser à une suite de coût $(val(t_n))_{n\in\mathbb{N}}$, où t_n est un n-calcul sur u. La fonction d'évaluation de u sera ainsi à valeur dans les suite de coûts. On formalise donc ici quelques notions sur de telles fonctions.

Soient $(E, \leq)$ et $(F, \leq)$ deux ensembles ordonnés. On dit qu'une fonction $f : E \to F^{\mathbb{N}}$ est *α-monotone* si

$$\forall a, b \in E, \quad a \leq b \to f(a) \preceq_\alpha f(b) \ .$$

En particulier, pour tout $a \in E$, $a \leq a$, donc $f(a) \preceq_\alpha f(a)$, d'où $f(a)$ est α-croissante.

On dira qu'une fonction est *monotone* si elle est α-monotone pour un certain α.

Si $f : E \to F^{\mathbb{N}}$ est α-monotone, on lui associe $\tilde{f} : E^{\mathbb{N}} \to F^{\mathbb{N}}$ définie par :

$$\text{pour tout } \vec{a} \in E^{\mathbb{N}} \text{ et } n \in \mathbb{N}, \qquad \tilde{f}(\vec{a})(n) = f(\vec{a}(n))(n) \ .$$

Le principe de $\tilde{f}$ est de pouvoir mimer le comportement de f, si l'argument est déjà une suite de E et non plus un élément unique.

Définition 2.2.8 *La relation $\sim_\alpha$ sur les fonctions $E \to F^{\mathbb{N}}$ est définie par $f \sim_\alpha g$ si pour tout $u \in E$, $f(u) \sim_\alpha g(u)$. Comme d'habitude, on définit aussi $\sim$ par $f \sim g$ s'il existe α tel que $f \sim_\alpha g$.*

On utilisera aussi ces notions avec l'ordre produit : si $(E, \leq)$ est un ensemble ordonné, l'ensembe E^* des mots sur E est ordonné de manière canonique par $a_1 \dots a_n \leq b_1 \dots b_m$ si et seulement si $m = n$ et $a_i \leq b_i$ pour tout $i \in [1, n]$.

On identifiera les éléments de $(E^{\mathbb{N}})^*$ (mots de suites) avec certains éléments de $(E^*)^{\mathbb{N}}$ (suites de mots de même longeur).

2.2.6 Idéaux d'un ensemble ordonné

Cette notion est essentielle pour définir la sémantique d'un semigroupe de stabilisation. En effet, l'ensemble des mots qui ont une grande valeur pour une fonction de coût f sera toujours représenté par un idéal dans le semigroupe de stabilisation reconnaissant f. Ceci correspond à l'idée intuitive selon laquelle la valeur de la fonction ne pourra qu'augmenter si certains évènements sont répétés un plus grand nombre de fois.

Définition 2.2.9 *Soit $(E, \leq)$ un ensemble ordonné, un* idéal *est un ensemble $I \subseteq E$, clos par $\leq$. C'est à-dire que pour tout $a \in I$ et $b \leq a$, on a $b \in I$.*

*Si $a \in E$, l'*idéal généré par *a est $I_a := \{b \in E \ : \ b \leq a\}$. C'est le plus petit idéal qui contient a.*

Si $\vec{a} \in E^{\mathbb{N}}$ et I est un idéal, on définit $I[\vec{a}] := \sup\{n+1 \ : \ \vec{a}(n) \in I\}$.[1] *Si I est un idéal, son complément dans E est noté $\overline{I}$. Si $\vec{a} \in E^{\mathbb{N}}$, on définit $\overline{I}[\vec{a}] = \inf\{n \ : \ \vec{a}(n) \in \overline{I}\}$.*

Proposition 2.2.10 ([Col09]) *Pour toutes suites $\vec{a}, \vec{b} \in E^{\mathbb{N}}$, les assertions suivantes sont équivalentes :*
- *$\vec{a} \preceq_\alpha \vec{b}$;*
- *pour tout idéal $I \subseteq E$, $\overline{I}[\vec{a}] \succcurlyeq_\alpha I[\vec{b}]$*
- *$\forall c \in E$, $\overline{I_c}[\vec{a}] \succcurlyeq_\alpha I_c[\vec{b}]$.*

Proposition 2.2.11 *Soit f et g des fonctions $E \to S^{\mathbb{N}}$ monotones telles que $f \sim g$. Alors pour tout idéal I de S, les fonctions de coût $f_I : u \mapsto I[f(u)]$ et $g_I : u \mapsto I[g(u)]$ sont équivalentes.*

Démonstration On suppose f α_f-monotone, g α_g-monotone, et $f \sim_\beta g$.

Soit $u \in E$, et $n = f_I(u) = I[f(u)]$. Alors $g(u)(\beta(n)) \geq f(u)(n) \notin I$. I est un idéal donc on obtient $g(u)(\beta(n)) \notin I$. $g(u)$ est α_g-croissante donc par la proposition 2.2.10, $I[g(u)] \leq \alpha_g(\beta(n))$. On a obtenu $g_I \preccurlyeq f_I$. Par symétrie, on obtient $f_I \approx g_I$. □

2.2.7 Sémantique des semigroupes de stabilisation

Grâce aux notions de suites de coût et d'idéaux introduites ci-dessus, on peut maintenant décrire précisément les fonctions de coût que peut reconnaître un semigroupe de stabilisation.

Le lemme suivant établit une première cohérence sur l'ensemble des calculs (toujours de hauteur H pour l'instant) :

Lemme 2.2.12 ([Col11]) *Pour tout $u \in S^+$, et pour toutes suites (t_n) et (t'_n) telles que chaque t_n et t'_n soient des n-calculs sur u, on a $(val(t_n))_{n\in\mathbb{N}} \sim (val(t'_n))_{n\in\mathbb{N}}$.*

Cela signifie que pour évaluer un mot u au moyen d'un n-calcul, le choix de ce calcul n'a pas d'importance (modulo $\sim$).

Le lemme 2.2.12 nous permet de définir une fonction d'évaluation $\rho : S^+ \to \mathbb{N} \to S$ par

$$\rho(u)(n) = val(t), \text{ où } t \text{ est un } n\text{-calcul quelconque sur } u.$$

La fonction ρ est dite *compatible* avec $\mathbf{S}$. Cette fonction peut être vue comme une généralisation du produit π, car si $\mathbf{S}$ est un semigroupe classique, on peut toujours prendre $\rho = \pi$. Elle est définie modulo $\sim$, car les valeurs précises peuvent dépendre du choix de t. On dira donc qu'une fonction ρ' est compatible avec $\mathbf{S}$ si $\rho' \sim \rho$.

1. Le $+1$ est justifié par une commodité de calcul, mais n'est pas strictement nécessaire.

On peut remarquer que ρ dépend ici de H. En fait prendre un H plus grand ne changerait pas la théorie, ceci correspondrait juste à appliquer une fonction de correction α à toutes les suites de coût et fonctions de coût. Plus précisément on dispose du lemme suivant :

Lemme 2.2.13 ([Col11]) *Pour tout k, il existe α tel que si ρ est définie relativement à H et ρ' relativement à $H' \geq H$, alors $\rho \sim_\alpha \rho'$.*

Ceci signifie que l'on peut par exemple regarder les arbres de calcul de hauteur $2H$, ou H^2, leur valeur décrira toujours une fonction compatible. L'important est que la hauteur maximale des arbres considérés doit être indépendante du mot donné en entrée, et au moins $3|S|$.

Le théorème suivant utilise les définitions de la section 2.2.5, en particulier pour $\tilde{\rho}$.

Théorème 2.2.14 ([Col09]) *Si ρ est une fonction compatible avec $\mathbf{S}$, alors il existe α tel que :*

Lettre. *pour tout $a \in S$, $\rho(a) \sim_\alpha a$,*

Produit. *pour tous $a, b \in S$, $\rho(ab) \sim_\alpha a \cdot b$,*

Stabilisation. *pour tout $e \in E(\mathbf{S})$, $m \in \mathbb{N}$, $\rho(e^m) \sim_\alpha (e^\sharp|_m e)$,*

Substitution. *pour tous $u_1, \ldots, u_n \in S^+$ et $n \in \mathbb{N}$,*
$\rho(u_1 \ldots u_n) \sim_\alpha \tilde{\rho}(\rho(u_1) \ldots \rho(u_n))$

Notons que ce théorème découle du Lemme 2.2.13, puisque chaque cas correspond à combiner des arbres de calcul :

Lettre découle de l'existence pour tout n du n-calcul trivial ayant pour seul noeud a.

Produit on lie deux arbres avec un noeud binaire, la hauteur augmente d'un.

Stabilisation on lie m arbres avec un noeud de stabilisation/idempotent, la hauteur augmente d'un.

Substitution on construit un arbre de hauteur H partant des racines des n arbres fournis, la hauteur peut doubler.

Ecrire des équations utilisant une fonction compatible ρ est donc un raccourci pour décrire des opérations effectuées sur les arbres de calcul.

On remarque qu'une telle fonction ρ peut être vue comme une généralisation du produit $\pi : S^+ \to S$. En effet, si $\sharp$ est l'identité, alors $\rho(u) = \pi(u)$ pour tout $u \in S^+$. La fonction ρ a pour but de faire un produit tout en gardant trace d'informations quantitatives.

Théorème 2.2.15 ([Col11]) *Si ρ' vérifie toutes les propriétés du théorème 2.2.14, alors $\rho' \sim \rho$. En d'autres termes, les fonctions compatibles sont caractérisées par le théorème 2.2.14.*

Ce théorème est fondamental, car il garantit qu'il n'y a qu'une manière (modulo $\sim$) d'évaluer un mot de S^+ au moyen d'une fonction compatible.

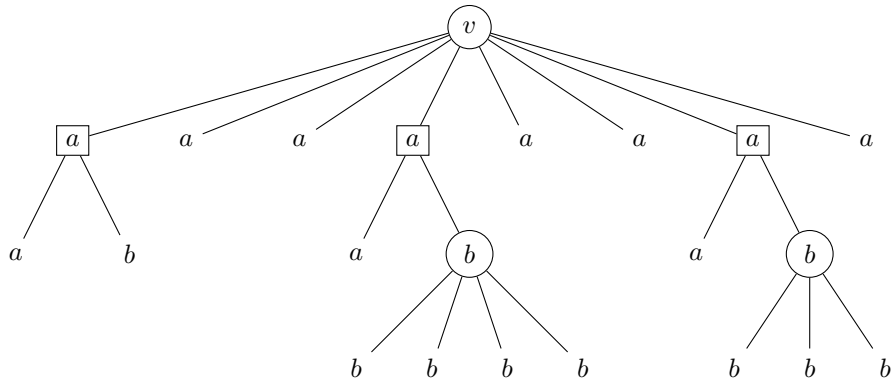
Remarque 2.2.16 *A l'origine, les fonctions compatibles étaient définies au moyen des propriétés du théorème 2.2.14, qui étaient alors des axiomes. La notion d'arbre de calcul a été introduite par Colcombet un peu plus tard. On verra que suivant l'utilisation que l'on veut en faire, il est utile d'avoir les deux formalismes à notre disposition.*

Exemple 2.2.17 *Soit S le semigroupe de stabilisation à 3 éléments $\bot \leq a \leq b$, dont le produit est défini par : $x \cdot y = \min_{\leq}(x, y)$ (b élément neutre), et la stabilisation par $b^\sharp = b$ et $a^\sharp = \bot^\sharp = \bot$. On définit $\rho : S^+ \to S^{\mathbb{N}}$ par :*

$$\rho(u) = \begin{cases} b & \text{si } u \in b^+ \\ \bot|_{|u|_a} a & \text{si } u \in b^*(ab^*)^+ \\ \bot & \text{sinon } (u \text{ contient } \bot). \end{cases}$$

Alors ρ est compatible avec S. On peut prouver ce fait en construisant pour tout u un n-calcul de hauteur 3 et de valeur $\rho(u)(n)$.

Par exemple pour $u = abaaabbbbaaabbba$, et $n \in \mathbb{N}$, on obtient le n-calcul suivant :



On remarque que le nombre de fils de la racine est exactement $|u|_a = 8$. Deux cas sont possibles :

- *$n \leq 8$: la racine est un noeud idempotent, et $v = a$.*
- *$n > 8$: la racine est un noeud de stabilisation, et $v = a^\sharp = \bot$.*

On a donc bien $\rho(u) = \bot|_{|u|_a} a$, dans le cas où $|u|_a \geq 1$.

On peut construire un arbre similaire pour les cas où $|u|_a = 0$. La valeur de n n'importera plus car b et $\bot$ sont stables par $\sharp$.

Le lemme suivant décrit plus précisément la relation entre ρ et le produit classique π.

Lemme 2.2.18 *Soit ρ compatible avec $\mathbf{S}$. Il existe γ tel que pour tout $n \in \mathbb{N}$ et $u \in S^+$, si $|u| \leq n$ alors pour tout $k \geq \gamma(n), \rho(u)(k) = \pi(u)$*

Démonstration Soit $u \in S^+$, et $k \geq |u|$. Soit ρ' une fonction compatible définie directement via les calculs. Soit t un k-calcul sur u. t ne peut pas contenir de noeud de stabilisation, car le nombre de fils est toujours borné par la longueur totale du mot (qui correspond au nombre de feuilles de t). Chaque noeud est donc étiqueté par le produit de ses fils, et donc par induction sur la hauteur de l'arbre, $val(t) = \pi(u)$.

Ceci montre que pour $k \geq |u|$, $\rho'(u)(k) = \pi(u)$. Finalement, si ρ est une fonction compatible quelconque, alors il existe α telle que $\rho \sim_\alpha \rho'$. On aura donc pour tout $k \geq \alpha(|u|)$, $\rho(u)(k) = \pi(u)$ □

Définition 2.2.19 ([Col11]) *Dans la suite, on utilisera aussi la notion de* n-sur-calcul *(resp.* n-sous-calcul*) : la définition est la même que pour un* n*-calcul, sauf que les étiquettes des noeuds parents doivent juste être supérieures (resp. inférieures) au résultat donné par leurs fils. Par exemple dans un sur-calcul, un noeud binaire* p *de fils* p_1 *et* p_2 *devra être étiqueté par* $t(p) \geq t(p_1) \cdot t(p_2)$*. De même, les feuilles de l'arbre doivent être supérieures (resp. inférieures) aus lettres du mot que l'on veut évaluer.*

On peut remarquer que les n-calculs sont exactement les arbres qui sont à la fois des n-sur-calculs et des n-sous-calculs.

2.2.8 Fonctions de coût reconnues

On dispose maintenant de tous les outils pour définir la sémantique d'un semigroupe de stabilisation, i.e. définir la fonction de coût qu'il reconnaît.

Soit $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ un semigroupe de stabilisation. Soit $h : \mathbb{A} \to S$ un morphisme, que l'on étend canoniquement à $h : \mathbb{A}^+ \to S^+$ (la longueur des mots est préservée). Soit $I \subseteq S$ un idéal de $\mathbf{S}$. Alors le triplet $\mathbf{S}, h, I$ *reconnaît* la fonction $f : \mathbb{A}^+ \to \mathbb{N}_\infty$, définie par

$$f(u) = I[\rho(h(u))] = \inf \{n \in \mathbb{N} : \rho(h(u))(n) \notin I\}.$$

Une fonction de coût : $\mathbb{A}^+ \to \mathbb{N}_\infty$ est *reconnaissable* si elle est équivalente (pour $\approx$) à une fonction reconnue par un triplet $\mathbf{S}, h, I$.

La propriété suivante assure l'unicité de la sémantique d'un semigroupe de stabilisation :

Proposition 2.2.20 ([Col09]) *Si* ρ *est une fonction monotone* $S^+ \to S^{\mathbb{N}}$*, soit* $f_\rho : \mathbb{A}^+ \to \mathbb{N}_\infty$ *définie par* $f_\rho(u) = I[\rho(h(u))]$*. Alors pour tous* $\rho \sim \rho'$*, on a* $f_\rho \approx f_{\rho'}$*.*

Ceci montre qu'il est cohérent de considérer les fonctions compatibles comme une $\sim$-classe, puisque tous les représentants de cette classe définissent la même fonction de coût.

Exemple 2.2.21 *Soit* $\mathbb{A} = \{a, b\}$*, la fonction* $u \mapsto |u|_a$ *est reconnaissable. On reprend le semigroupe de stabilisation de l'exemple 2.2.17. On pose* $h(a) =$

$a, h(b) = b$, *et* $I = \{\perp\}$. *On obtient ainsi* $|u|_a = \inf\{n \in \mathbb{N} : \rho(h(u))(n) \neq \perp\}$ *pour tout* $u \in \mathbb{A}^+$.

Remarque 2.2.22 *Si* $\cdot^\sharp$ *est l'identité, alors* $\mathbf{S}$ *est un semigroupe classique, et le produit généralisé* $\pi : S^+ \to S$ *est compatible avec* $\mathbf{S}$. *Ainsi, si* $P \subseteq S$ *et* $L = \pi^-1(h^-1(P))$ *est un langage reconnu par* $\mathbf{S}$, *alors* χ_L *est reconnu par* $\mathbf{S}, h, I$ *avec* $I = S \setminus P$.

Proposition 2.2.23 ([Col11]) *Soit* f *reconnue par un triplet* $\mathbf{S}, h, I$. *Pour tout* k *il existe* α *tel que pour tout* $u \in \mathbb{A}^+$ *et* $n \in \mathbb{N}$, *si* t *est un* n*-sous-calcul sur* $h(u)$ *de valeur* $a \notin I$ *et de profondeur au plus* k, *alors* $f(u) \leq_\alpha n$.

Réciproquement, si t *est un* n*-sur-calcul sur* $h(u)$ *de valeur* $a \in I$ *et de profondeur au plus* k, *alors* $f(u) \geq_\alpha n$.

Le résultat suivant est fondamental, et montre que la notion de régularité a un sens pour les fonctions de coût :

Théorème 2.2.24 ([Col09]) *Il est effectivement équivalent pour une fonction de coût d'être reconnue par :*
- *un semigroupe de stabilisation,*
- *un* B*-automate,*
- *un* S*-automate.*

On dira qu'une telle fonction de coût est régulière.

2.2.9 Opérations sur les semigroupes de stabilisation

Définition 2.2.25 (Produit) *Soit* $\mathbf{S}_1 = \langle S_1, \cdot_1, \leq_1, \sharp_1\rangle$ *et* $\mathbf{S}_2 = \langle S_2, \cdot_2, \leq_2, \sharp_2\rangle$ *deux semigroupes de stabilisation. On définit leur produit* $\mathbf{S}_1 \times \mathbf{S}_2$ *par le tuple* $\langle S_1 \times S_2, \cdot, \leq, \sharp\rangle$ *avec* $(a_1, a_2)\cdot(b_1, b_2) = (a_1\cdot b_1, a_2\cdot b_2)$, $(a_1, a_2) \leq (b_1, b_2)$ *si* $a_1 \leq b_1$ *et* $a_2 \leq b_2$, *et* $(e_1, e_2)^\sharp = (e_1^{\sharp_1}, e_2^{\sharp_2})$.

Proposition 2.2.26 *Si* $\mathbf{S}_1$ *et* $\mathbf{S}_2$ *sont des semigroupes de stabilisation, alors* $\mathbf{S}_1 \times \mathbf{S}_2$ *en est aussi un.*

De plus, si ρ_1 *est compatible avec* $\mathbf{S}_1$ *et* ρ_2 *avec* $\mathbf{S}_2$ *alors* $\rho = (\rho_1, \rho_2)$ *est compatible avec* $\mathbf{S}_1 \times \mathbf{S}_2$.

Définition 2.2.27 (Morphisme) *Une fonction* ϕ *de* $\mathbf{S}$ *vers* $\mathbf{S}'$ *est un morphisme de semigroupes de stabilisation si*
- ϕ *est un morphisme de semigroupes ordonnés,*
- *pour tout* $e \in E(\mathbf{S})$, $\phi(e^\sharp) = \phi(e)^\sharp$.

Lemme 2.2.28 *Soit* $\mathbf{S}, \mathbf{S}'$ *deux semigroupes de stabilisation,* ρ *et* ρ' *compatibles avec* $\mathbf{S}$ *et* $\mathbf{S}'$. *On suppose qu'il existe un morphisme de semigroupes de stabilisation* τ *de* $\mathbf{S}$ *vers* $\mathbf{S}'$. *Soit* $\tau^+ : \mathbf{S}^+ \to \mathbf{S}'^+$ *et* $\tau^\mathbb{N} : \mathbf{S}^\mathbb{N} \to \mathbf{S}'^\mathbb{N}$ *les extensions canoniques de* τ *aux mots et aux suites. Alors* $\tau^\mathbb{N} \circ \rho \sim \rho' \circ \tau^+$.

Démonstration Il suffit de remarquer que si l'on applique τ aux étiquettes d'un n-calcul pour $\mathbf{S}$, on obtient un n-calcul pour $\mathbf{S}'$. En effet, les propriétés des calculs sont toutes préservées par morphisme de semigroupe de stabilisation.

Soit $u \in \mathbb{N}$ et $n \in \mathbb{N}$, $\rho(u)(n)$ est la valeur d'un n-calcul sur u. Donc $\tau(\rho(u)(n))$ est la valeur d'un n-calcul sur $\tau^+(u)$.

C'est vrai pour tout u et n, mais ρ et ρ' sont définies modulo $\sim$, donc $\tau^{\mathbb{N}} \circ \rho \sim \rho' \circ \tau^+$.

Remarque 2.2.29 *Cette preuve est un bon exemple de cas où la notion d'arbre de calcul est utile. On peut prouver ce résultat en utilisant seulement la caractérisation donnée par le Théorème 2.2.14, mais ceci résulte en une preuve beaucoup moins élégante et lisible.*

□

2.3 $\omega\sharp$-expressions et congruence syntactique

2.3.1 Congruence syntactique classique

Dans la théorie classique, le semigroupe minimal d'un langage peut être décrit via la relation de Myhill-Nerode. Le principe de cette relation, définie par rapport à un langage L, est d'identifier des mots u et v si L ne peut les distinguer dans aucun contexte. Plus formellement, $u \sim_L v$ si pour tous x, y de $\mathbb{A}^*$, $xuy \in L \Leftrightarrow xvy \in L$. Un langage L est régulier si et seulement si $\mathbb{A}^*/\sim_L$ est fini. De plus, dans ce cas, $\mathbb{A}^*/\sim_L$ est le semigroupe minimal reconnaissant L : il divise tout semigroupe reconnaissant L.

On vise ici à étendre cette définition aux fonctions de coût, et obtenir une caractérisation similaire. Pour cela il faut inclure des informations de nature quantitative dans les éléments et dans les contextes : on veut pouvoir exprimer des propriétés comme « les mots contenant beaucoup de a se comportent de la même manière que les mots contenant peu de a ».

C'est pourquoi on a besoin d'annoter les mots avec des informations supplémentaires portant sur certaines quantités. On nomme ces « mots enrichis » les $\omega\sharp$-expressions. Cette notion est une variante de la notion de $\sharp$-expression introduite par Hashigushi [Has90], qui était moins adaptée à la théorie des semigroupes de stabilisation. On commence par décrire les $\sharp$-expressions, puis on définira les $\omega\sharp$-expressions.

2.3.2 $\sharp$-expressions

On commence par définir les $\sharp$-expressions comme dans [Has90], en ajoutant juste un opérateur $\sharp$ aux mots pour signifier que certains facteurs (possiblement imbriqués) sont répétés « un grand nombre de fois ». Ceci diffère du $\sharp$ des semigroupes de stabilisation, où cet opérateur est restreint à un certain type d'éléments : les idempotents.

L'ensemble Expr des $\sharp$-expressions sur un alphabet $\mathbb{A}$ est donc défini comme suit :

$$e := a \in \mathbb{A} \mid ee' \mid e^\sharp$$

Si $e, e' \in \text{Expr}$ et $n \in \mathbb{N}$, on définit $e(n)$ comme le mot sur $\mathbb{A}$ obtenu en remplaçant dans e chaque occurence de $\sharp$ par un exposant n dans e, ce que l'on note $e[\sharp \leftarrow n]$. Par exemple si $e = a(a^\sharp b)^\sharp$, alors $e(3) = aaaabaaabaaab \in \mathbb{A}^+$.

Si l'on se fixe un semigroupe de stabilisation $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ et une fonction $h : \mathbb{A} \to S$, on peut définir l'évaluation d'une $\sharp$-expression dans $\mathbf{S}$ de la manière suivante :

- $\text{eval}(a) = h(a)$,
- $\text{eval}(ee') = \text{eval}(e) \cdot \text{eval}(e')$,
- si $\text{eval}(e)$ est idempotent, $\text{eval}(e^\sharp) = \text{eval}(e)^\sharp$.

On remarque que la fonction eval est seulement partielle : si $\text{eval}(e)$ est défini mais pas idempotent dans $\mathbf{S}$, alors $\text{eval}(e^\sharp)$ n'est pas défini.

On dira que e est *bien formée* pour $\mathbf{S}$ si $\text{eval}(e)$ est défini. Ceci signifie que l'on a utilisé $\sharp$ dans e seulement sur des facteurs qui s'évaluent en un idempotent dans $\mathbf{S}$.

Exemple 2.3.1 *Soit f la fonction de coût sur $\{a\}^*$ définie par*

$$f(a^n) = \begin{cases} n & \text{si } n \text{ pair} \\ 0 & \text{sinon} \end{cases}$$

Alors f est reconnue par $\mathbf{S}, h, I$, où $\mathbf{S} = \langle S, \cdot, leq, \sharp \rangle$ est décrit ci-dessous. On a tout d'abord $S = \{a, aa, (aa)^\sharp, (aa)^\sharp a\}$, $I = \{(aa)^\sharp\}$, et $h(a) = a$. Le produit et la stabilisation sont lisibles sur les éléments, et on ajoute les règles $aa \cdot a = a$ et $(aa)^\sharp a \cdot a = (aa)^\sharp$. La $\sharp$-expression $aaa(aa)^\sharp$ est bien formée pour S mais la $\sharp$-expression $a^\sharp$ ne l'est pas : a n'est pas idempotent.

Les $\sharp$-expressions mal formées pour $\mathbf{S}$ ont un comportement hybride, qui ne reflète pas d'élément particulier de $\mathbf{S}$: dans l'exemple ci-dessus $\rho(a^\sharp(n))(k)$ va en fait osciller entre deux éléments de S. Plus formellement, pour tout k fixé et quand $n \to \infty$, $\rho(a^\sharp(n))(k)$ a deux valeurs d'adhérence : $(aa)^\sharp$ et $(aa)^\sharp a$. On voudrait donc retirer les $\sharp$-expressions mal formées, avant de quotienter l'ensemble des $\sharp$-expressions pour obtenir le semigroupe de stabilisation minimal de f.

2.3.3 $\omega\sharp$-expressions

La définition et l'étude des $\omega\sharp$-expressions sont effectuées dans [Kup11].

Comme auparavant, on veut spécifier quelles parties des mots sont répétées « un grand nombre de fois », au moyen d'un constructeur $\cdot^\sharp$.

Cependant, pour rester cohérent avec les notions algébriques introduites, on ne veut appliquer cette opération qu'à des éléments idempotents. C'est pourquoi on introduit également un constructeur $\cdot^\omega$, qui permet de spécifier qu'on commence par associer un idempotent à l'expression donnée en argument.

En résumé, la syntaxe correcte pour les $\omega\sharp$-expressions est donnée par

$$e := a \in \mathbb{A} | ee | e^{\omega} | e^{\omega\sharp}$$

Soit Oexpr l'ensemble des $\omega\sharp$-expressions ainsi formées, sur l'alphabet $\mathbb{A}$ fixé.

Un *contexte* $C[x]$ est une $\omega\sharp$-expression contenant des occurences de la variable libre x. Si e est une $\omega\sharp$-expression, $C[E]$ est l'$\omega\sharp$-expression obtenue en replaçant toutes les occurences de x par e dans $C[x]$, c'est-à-dire $C[e] = C[x][x \leftarrow e]$. Soit $\mathrm{C_{OE}}$ l'ensemble des contextes sur l'alphabet $\mathbb{A}$ fixé. De manière équivalente, on peut définir $\mathrm{C_{OE}}$ comme étant l'ensemble des $\omega\sharp$-expressions sur l'alphabet $\mathbb{A} \cup \{x\}$.

Notons que des notions similaires sont définies par Toruńczyk dans sa thèse [Tor11], en se basant sur les mots profinis. Ces travaux ont été effectués de manière indépendante, et leurs formalismes sont différents (en particulier on ne mentionnera pas les mots profinis ici), mais certaines des notions sont communes.

2.3.4 Sémantique

On peut maintenant définir formellement la sémantique des opérateurs ω et $\sharp$, afin d'utiliser l'ensemble Oexpr comme base pour la congruence syntactique des fonctions de coût.

Définition 2.3.2 *Si $e \in$ Oexpr et $k, n \in \mathbb{N}$, on définit $e(k, n)$ comme le mot $e[\omega \leftarrow k, \sharp \leftarrow n]$, où l'exponentiation par un entier dénote la concaténation itérée.*

Lemme 2.3.3 *Soit f une fonction de coût reconnue par un semigroupe de stabilisation. Alors il existe $K_f \in \mathbb{N}$ tel que pour tout $e \in$ Oexpr, on se trouve dans l'un de ces deux cas :*

1. *$\forall k \geq K_f, \{f(e(k!, n)), n \in \mathbb{N}\}$ est borné : on dit alors que $e \in f^B$.*
2. *$\forall k \geq K_f, \lim_{n\to\infty} f(e(k!, n)) = \infty$: on dit alors que $e \in f^{\infty}$.*

Démonstration Soit f une fonction de coût reconnue par $\mathbf{S}, h, I$, avec $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$. On pose $K_f = |S|$.

On définit eval : Oexpr $\to S$ par induction :

- $\mathrm{eval}(a) = h(a)$ pour $a \in \mathbb{A}$,
- $\mathrm{eval}(e_1 e_2) = e_1 \cdot e_2$,
- $\mathrm{eval}(e^{\omega}) = e^{K_f!}$,
- $\mathrm{eval}(e^{\omega\sharp}) = \mathrm{eval}(e^{\omega})^{\sharp}$.

Autrement dit $\mathrm{eval}(e) = \mathrm{eval}(e[\omega \leftarrow K_f!])$, où l'évaluation sur les $\sharp$-expressions est définie dans la Section 2.3.2. On peut remarquer que pour tout $e \in$ Oexpr, $\mathrm{eval}(e^{\omega})$ est un idempotent (résultat classique sur les semigroupes : $K_f!$ est forcément un multiple de la puissance idempotente de $\mathrm{eval}(e)$), ce qui entraine $e[\omega \leftarrow K_f!]$ bien formée pour S, et donc $\mathrm{eval}(e^{\omega\sharp})$ est bien défini. De plus, la fonction eval ne change pas si on remplace K_f par un entier plus grand dans sa définition.

On définit aussi $\lim(e)$ de la même manière que pour eval, sauf que $\lim(e^{\omega\sharp}) = \lim(e^\omega)$: les $\sharp$ sont ignorés. Il est facile de montrer par induction que pour tout e, $\mathrm{eval}(e) \leq \lim(e)$.

Pour tout $n \in \mathbb{N}$, on pose $e(n) := e(K_f!, n)$ où les lettres a sont remplacées par $h(a)$, leur image dans S. $e(n)$ est un mot de S^+, et on s'intéresse au comportement de $f(e(n))$ quand n tend vers l'infini.

Par le Lemme 2.2.18, on sait qu'il existe $\gamma : \mathbb{N} \to \mathbb{N}$ tel que pour tout $n \in \mathbb{N}$, pour tout $m \geq \gamma(n)$, $\rho(e(n))(m) = \pi(e(n))$. Or, les puissances n sont toujours appliquées à un idempotent, donc pour tout $n \in \mathbb{N}$, $\pi(e(n)) = \pi(u_1) = \lim(e)$.

On va en fait montrer un résultat plus fort :

Lemme 2.3.4 *Il existe α_e (dépendant de e) tel que pour tout $n \in \mathbb{N}$, $\rho(e(n)) \sim_\beta \mathrm{eval}(e)|_n \lim(e)$.*

Démonstration du Lemme 2.3.4

On procède par induction sur e : on suppose que le résultat est vrai pour les sous-expressions de e, et on construit pour tout m un m-calcul sur $e(n)$ témoignant du résultat.

- si $e = a \in \mathbb{A}$, tout calcul est un singleton, et $\mathrm{eval}(e) = \lim(e) = a$, donc le résultat est vrai.
- si $e = rs$, alors $\rho(e(n)) = \rho(r(n)s(n)) \sim_\beta \tilde{\rho}[\rho(r(n))\rho(s(n)]$, mais par hypothèse de récurrence, il existe α_r et α_s satisfaisant le lemme pour r et s. On peut donc choisir $\alpha_e = \beta \circ \max(\alpha_r, \alpha_s)$ pour obtenir le résultat
- si $e = r^\sharp$ avec $r = r'^\omega$, alors

$$\rho(e(n)) \sim_\beta \tilde{\rho}(\rho(r(n))^n) \sim_{\alpha_r} \tilde{\rho}((\mathrm{eval}(r)|_n \lim(r))^n).$$

  Or, $\mathrm{eval}(r)$ est idempotent, donc par le Théorème 2.2.14 on sait que $\rho(\mathrm{eval}(r)^n) \sim_\beta \mathrm{eval}(r)^\sharp|_n \mathrm{eval}(r)$. De plus, par le Lemme 2.2.18, pour $k > \gamma(n)$, $\rho(e(n))(k) = \pi(e(n))$.
  Finalement, il existe α (obtenue par composition de α_r, β, γ) telle que pour $\alpha(k) \leq n$, $\rho(e(n)) = \mathrm{eval}(r)^\sharp = \mathrm{eval}(e)$, et pour $k \geq \alpha(n)$, $\rho(e(n)) = \pi(e(n)) = \lim(e)$. Cela signifie que $\rho(e(n)) \sim_\alpha \mathrm{eval}(e)|_n \lim(e)$.

□

Avec ce lemme, on peut conclure la preuve du Lemme 2.3.3 en examinant les cas suivants :

- Si $\lim(e) \in I$, alors $\mathrm{eval}(e) \in I$ car I est un idéal et $\mathrm{eval}(e) \leq \lim(e)$. Pour tout n, tous les éléments de $\rho(e(n))$ sont donc dans I, et pour tout $k \geq K_f$, $f(e(k!, n)) = \infty$. On a donc $e \in f^\infty$.
- Si $\lim(e) \notin I$ et $\mathrm{eval}(e) \in I$, alors $f(e(k!, n)) \approx_\alpha n$. On a donc $e \in f^\infty$.
- Si $\lim(e) \notin I$ et $\mathrm{eval}(e) \notin I$, alors $f(e(k!, n)) \approx_\alpha 0$. On a donc $e \in f^B$.

□

Remarquons que l'on a montré le lemme suivant :

Lemme 2.3.5 *Si f est reconnue par $\mathbf{S}, h, I$, et* eval *est la fonction d'évaluation relative à $\mathbf{S}, h$, alors pour tout $e \in$* Oexpr *:*

$$e \in f^B \Leftrightarrow \mathrm{eval}(e) \notin I$$

Ici, f^B et f^∞ sont les analogues pour les fonctions de coûts des propriétés « être dans L » et « ne pas être dans L » de la théorie des langages. Cette notion est maintenant asymptotique, car seulement le comportement à l'infini (borné ou non) nous intéresse dans le cas des fonctions de coût. Le Lemme 2.3.5 étend la définition de reconnaissance d'un langage par semigroupe classique. En effet, comme décrit dans la section 2.2.1, pour voir si un mot appartient à un langage caractérisé par une partie P d'un semigroupe $\mathbf{S}$, on évalue ce mot dans $\mathbf{S}$ et on regarde si le résultat est dans P. Ici, les mots sont remplacés par des $\omega\sharp$-expressions, puisque le comportement d'une fonction de coût peut dépendre de données quantitatives, décrites par l'opérateur $\sharp$. On évalue ensuite les $\omega\sharp$-expressions dans le semigroupe de stabilisation, et on regarde si le résultat est dans I ou non. Remarquons que sans opérateur $\sharp$ dans les $\omega\sharp$-expressions, les suites définies sont constantes, et on retrouve la théorie des langages (il faut complémenter I pour obtenir la partie acceptante).

Contrairement au cas des langages, où l'appartenance d'un mot au langage peut être définie même pour les langages non réguliers, ici f^∞ et f^B sont définies uniquement si f est régulière. Sans cela, on ne peut pas définir la sémantique de l'opérateur ω, puisqu'il effectue la puissance idempotente dans un semigroupe particulier. Plus précisément, la constante K_f définie dans l'énoncé du Lemme 2.3.3 peut ne pas exister si f n'est pas régulière.

2.3.5 Congruence syntactique pour les fonctions de coût

Définition 2.3.6 *Soit f une fonction de coût régulière, on note $e \rightleftharpoons_f e'$ si $(e \in f^B \Leftrightarrow\ e' \in f^B)$. On définit finalement*

$$e \equiv_f e' \text{ si } \forall C[x] \in \mathrm{C_{OE}}, C[e] \rightleftharpoons_f C[e']$$

Remarque 2.3.7 *Si $u, v \in \mathbb{A}^*$, et L est un langage régulier, alors $u \sim_L v$ si et seulement si $u \equiv_{\chi_L} v$ ($\sim_L$ étant la congruence syntactique de L). En ce sens, $\equiv$ étend la congruence syntactique des langages.*

Définition 2.3.8 *Soit f une fonction de coût régulière, et $\mathbf{S}$ un semigroupe de stabilisation. On dit que $\mathbf{S}$ est le* semigroupe de stabilisation minimal *de f s'il divise tout semigroupe de stabilisation reconnaissant f. Cela signifie que tout $\mathbf{S}'$ reconnaissant f possède un sous-semigroupe de stabilisation $\mathbf{S}''$ tel qu'il existe un morphisme surjectif de semigroupes de stabilisation $\pi : \mathbf{S}'' \to \mathbf{S}$.*

On va montrer que $\equiv_f$ est une congruence syntactique sur Oexpr, c'est-à-dire que Oexpr$/\equiv_f$ est le *semigroupe de stabilisation minimal de f*.

Si f est une fonction de coût régulière, soit $S_f =$ Oexpr$/\equiv_f$, l'ensemble des classes d'équivalence de Oexpr pour la relation $\equiv_f$.

Lemme 2.3.9 *On peut munir S_f d'une structure de semigroupe de stabilisation $\mathbf{S}_f = \langle S_f, \cdot, \leq, \sharp \rangle$.*

Démonstration On note $\overline{e}$ la classe d'équivalence d'une $\omega\sharp$-expressions pour la relation $\equiv_f$.

On définit $\cdot$ et $\sharp$ sur S_f de manière naturelle, par $\overline{e_1} \cdot \overline{e_2} = \overline{e_1 e_2}$, et si $\overline{e}$ idempotent, $\overline{e}^\sharp = \overline{e^{\omega\sharp}}$.

Il faut montrer que ces opérations sont bien définies, c'est-à-dire que le résultat ne dépend pas du choix des représentants.

Soit $e_1' \equiv_f e_1$. On veut montrer que $\overline{e_1 e_2} = \overline{e_1' e_2}$ (c'est suffisant par symétrie). Soit $C[x]$ un contexte, et $C'[x] = C[xe_2]$. Alors $C'[e_1] \rightleftharpoons_f C'[e_1']$ par définition de $\equiv_f$. Donc $C[e_1 e_2] = C[e_1' e_2]$. C'est vrai pour tout contexte $C[x]$ donc on a bien $e_1 e_2 \equiv_f e_1' e_2$.

Soit $e' \equiv_f e$, avec $\overline{e}$ idempotent pour $\cdot$. On veut montrer que $e^{\omega\sharp} \equiv_f e'^{\omega\sharp}$. Soit $C[x]$ un contexte, et $C'[x] = C[x^{\omega\sharp}]$. Alors $C'[e] \rightleftharpoons_f C'[e']$ par définition de $\equiv_f$. Donc $C[e^{\omega\sharp}] = C[e'^{\omega\sharp}]$. C'est vrai pour tout contexte $C[x]$ donc on a bien $e^{\omega\sharp} \equiv_f e'^{\omega\sharp}$.

Finalement, on peut définir $\leq$ comme étant le plus petit ordre satisfaisant les axiomes des semigroupes de stabilisation, c'est-à-dire compatible avec $\cdot$ et $\sharp$, et tel que pour tout $s \in E(S_f)$, $s^\sharp \leq s$.

□

Théorème 2.3.10 $\mathbf{S}_f$ *est le semigroupe de stabilisation minimal de f.*

Démonstration On montre que $\mathbf{S}_f$ est quotient d'un sous-semigroupe de stabilisation de tout semigroupe de stabilisation reconnaissant f.

Soit $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$, $h : \mathbb{A} \to S$ et I un idéal de $\mathbf{S}$ tel que $\mathbf{S}, h, I$ reconnaît f.

Soit $\langle h(\mathbb{A}) \rangle^\sharp$ la clôture de $h(\mathbb{A})$ par produit et stabilisation dans S.

Quitte à considérer seulement un sous-semigroupe de stabilisation de $\mathbf{S}$, on peut supposer que $\langle h(\mathbb{A}) \rangle^\sharp = S$.

On reprend la fonction eval de la preuve du Lemme 2.3.3, définie par rapport à $\mathbf{S}, h$. On peut remarquer que eval : Oexpr $\to S$ est maintenant surjective.

Lemme 2.3.11 *Si $e, e' \in$ Oexpr sont tels que* $\mathrm{eval}(e) = \mathrm{eval}(e')$*, alors $e \equiv_f e'$.*

Démonstration du Lemme 2.3.11 Soit $C[x]$ un contexte, on veut montrer que $C[e] \rightleftharpoons_f C[e']$.

Par induction sur $C[x]$, il est facile de voir que $\mathrm{eval}(C[e]) = \mathrm{eval}(C[e'])$.

Or, en appliquant le Lemme 2.3.5 à $C[e]$ et $C[e']$, on obtient $C[e] \in f^B \Leftrightarrow \mathrm{eval}(C[e]) \notin I \Leftrightarrow \mathrm{eval}(C[e']) \notin I \Leftrightarrow C[e'] \in f^B$. On a donc montré $C[e] \rightleftharpoons_f C[e']$, et donc $e \equiv_f e'$. □

On peut maintenant définir un morphisme surjectif de semigroupes de stabilisation $\phi : \mathbf{S} \to \mathbf{S}_f$ par $\phi(\mathrm{eval}(e)) = \overline{e}$. On définit également la relation associée $\equiv$ sur S par $x \equiv y$ si $\phi(x) = \phi(y)$. On peut donc remarquer que $\mathbf{S}_f$ est isomorphe à $\mathbf{S}/\equiv$.

Il reste à montrer que $\mathbf{S}_f$ reconnaît bien f. Soit $I_f = \phi(I)$, $h_f = \phi^+ \circ h$, et ρ_f compatible avec $\mathbf{S}_f$. Par le Lemme 2.2.28, $\phi^{\mathbb{N}} \circ \rho \sim \rho_f \circ \phi^+$. Ceci signifie qu'il existe α tel que pour tout $u \in S^+$, $\phi^{\mathbb{N}}(\rho(u)) \sim_\alpha \rho_f(\phi^+(u))$.

En particulier, pour tout $u \in \mathbb{A}^+$, $\phi^{\mathbb{N}}(\rho(h(u))) \sim_\alpha \rho_f(\phi^+(h(u)))$, et donc $I_f[\phi^{\mathbb{N}}(\rho(h(u)))] \approx_\alpha I_f[\rho_f(\phi^+(h(u)))]$.

Mais $I_f = \phi(I)$, donc $I_f[\phi^{\mathbb{N}}(\rho(h(u)))] = I[\rho(h(u))]$. Finalement, on a bien $I_f[\rho_f(h_f(u))] \approx_\alpha I[\rho(h(u))] = f(u)$ c'est-à-dire $\mathbf{S}_f, h_f, I_f$ reconnaît f.

□

2.3.6 Procédure de minimisation

Si l'on dispose d'un semigroupe de stabilisation $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ qui reconnait f, on voudrait pouvoir obtenir $\mathbf{S}_f$ à partir de $\mathbf{S}$ de manière effective, notamment pour pouvoir ensuite décider des propriétés de f par l'étude de $\mathbf{S}_f$.

Lemme 2.3.12 *La relation $\equiv$, définie dans la preuve du Théorème 2.3.10 par* $\text{eval}(e) \equiv \text{eval}(e')$ *si $e \equiv_f e'$, est la plus grande relation sur $S = \langle h(\mathbb{A}) \rangle^\sharp$ vérifiant pour tout $x, y \in S$:*

$$(Prop) \begin{cases} (1)\ x \equiv y \Rightarrow (x \in I \Leftrightarrow y \in I) \\ (2)\ x \equiv y \Rightarrow \forall a \in S, a \cdot x \equiv a \cdot y \\ (3)\ x \equiv y \Rightarrow \forall a \in S, x \cdot a \equiv y \cdot a \\ (4)\ x \equiv y \Rightarrow x^\sharp \equiv y^\sharp \end{cases}$$

Démonstration On montre d'abord que $\equiv$ vérifie (*Prop*) : les propriétés (2), (3) et (4) sont garanties par le fait que ϕ est un morphisme de semigroupes de stabilisation. La propriété (1) se déduit du fait que pour tout $\omega\sharp$-expression e, $\text{eval}(e) \in I \Leftrightarrow e \in f^\infty$. En conséquence, si $x = \text{eval}(e_x)$ et $y = \text{eval}(e_y)$ sont tels que $e_x \equiv_f e_y$, alors $x \in I \Leftrightarrow e_x \in f^\infty \Leftrightarrow e_y \in f^\infty \Leftrightarrow y \in I$.

On montre maintenant que c'est la plus grande relation vérifiant (*Prop*). On suppose qu'il existe une relation $\asymp$ vérifiant (*Prop*), et $a \asymp b$ tels que $a \not\equiv b$.

Cela signifie qu'il existe $e_a, e_b \in \text{Oexpr}$ avec $a = \text{eval}(e_a)$, $b = \text{eval}(e_b)$, et $e_a \not\equiv_f e_b$. Il existe donc un contexte $C[x]$ avec par exemple $C[e_a] \in f^B$ et $C[e_b] \in f^\infty$. Cela est équivalent à $\text{eval}(C[e_a]) \notin I$ et $\text{eval}(C[e_b]) \in I$. Or, les propriétés (*Prop*) sont préservées par produit et stabilisation donc $\text{eval}(e_a) \asymp \text{eval}(e_b) \implies \text{eval}(C[e_a]) \asymp \text{eval}(C[e_b])$. Ceci contredit la propriété (1), c'est absurde. La relation $\equiv$ est donc bien la plus grande relation vérifiant (*Prop*). □

Corollaire 2.3.13 *On peut minimiser un semigroupe de stabilisation en temps polynomial.*

Démonstration On commence par générer le sous-semigroupe de stabilisation engendré par $h(\mathbb{A})$. Cela nous permet de retirer les éléments inaccessibles de S. Cette procédure est quadratique en $|S|$ (il suffit de tester tous les produits et les $\sharp$ à disposition, jusqu'à ce qu'on ne génère plus de nouvel élément), et nous permet de supposer dorénavant $S = \langle h(\mathbb{A}) \rangle^\sharp$.

On va ensuite décrire un algorithme nous permettant de définir une relation d'équivalence sur S.

Initialement, on pose $\equiv = S \times S$: tous les éléments sont en relation.

On va ensuite itérativement retirer les couples (x, y) qui ne vérifient pas $(Prop)$ de la relation $\equiv$.

On réitère cette opération jusqu'à ce que la relation $\equiv$ vérifie $(Prop)$ pour tous $x, y \in S$. On obtient ainsi la plus grande relation vérifiant $(Prop)$. Le nombre d'étapes est en $O(|S|^2)$, puisqu'on retire au moins un couple à chaque étape, et les conditions (2) et (3) demandent un temps de vérification en $O(|S|)$, donc l'algorithme a une complexité $O(|S|^3)$.

Cet algorithme est un calcul de plus grand point fixe : soit $T = 2^{S^2}$ le treillis des parties de S^2, ordonné par l'inclusion. Soit $g : T \to T$ la fonction qui supprime les couples ne vérifiant pas $(Prop)$ dans un ensemble de couples X donné un argument. Alors la fonction g est monotone pour l'inclusion : si $X \subseteq Y$, alors tout couple supprimé dans Y le sera aussi dans X (s'il est présent), et par conséquent $g(X) \subseteq g(Y)$. Le théorème du point fixe de Kleene nous assure que l'algorithme calcule le plus grand point fixe de g, c'est -à-dire la plus grande relation vérifiant $(Prop)$. Par le Lemme 2.3.12, cette relation est celle utilisée précédemment pour obtenir $\mathbf{S}_f$ comme $\mathbf{S}/\equiv$. On a donc obtenu une description de $\mathbf{S}_f$, en identifiant les éléments équivalents pour $\equiv$. □

2.3.7 Cas des fonctions de coût non régulières

Dans le cas des langages, la congruence syntactique est encore bien définie, et présente toujours un nombre infini de classes d'équivalence si le langage n'est pas régulier.

On a vu que dans le cas de fonctions de coût, il faut disposer d'un semigroupe de stabilisation fini reconnaissant f pour définir la sémantique de l'opérateur ω, et donc la congruence syntactique. Ce fait représente une première différence avec les langages.

On va voir que la différence se situe à un niveau plus profond que ce point technique.

On pourrait oublier la contrainte selon laquelle $\sharp$ se définit uniquement sur les idempotents. Ceci reviendrait à définir l'ensemble Expr des $\sharp$-expressions (tel qu'originellement défini par Hashiguchi [Has90]) comme

$$e := a \in \mathbb{A} \mid ee \mid e^{\sharp}.$$

On pourrait ensuite définir pour toute fonction de coût f une relation $\sim_f$ sur les expressions : $e \sim_f e'$ si pour tout contexte $C[x]$ sur les $\sharp$-expressions, $\{f(C[e])(n), n \in \mathbb{N}\}$ est borné si et seulement si $\{f(C[e'])(n), n \in \mathbb{N}\}$ est borné. En particulier, si f est régulière reconnue par $\mathbf{S}$, la relation $\sim_f$ restreinte aux $\sharp$-expressions bien formées pour $\mathbf{S}$ coïncide avec $\equiv_f$. Par contre, on peut alors avoir des classes d'équivalence de $\sim_f$ qui ne correspondent pas à des éléments de $\mathbf{S}$.

Exemple 2.3.14 *Sur* $\mathbb{A} = \{a\}$*, soit* $f(u) = \begin{cases} |u| & \text{si } |u| \text{ pair} \\ 0 & \text{sinon} \end{cases}$

Alors le semigroupe de f *a 4 éléments :* a*,* aa*,* $(aa)^\sharp$ *et* $(aa)^\sharp a$*, avec* $I = \{(aa)^\sharp\}$*. La* $\sharp$*-expression* $a^\sharp$ *ne correspond à aucun de ces éléments : la cause en est qu'elle n'appartient ni à* f^B*, ni à* f^∞*, car la suite* $f(e(n))$ *a deux valeurs d'adhérence :* 0 *et* ∞*.*

En général, $\mathrm{Expr}/\sim_f$ a donc plus d'éléments que $\mathrm{Oexpr}/\equiv_f$ (ce dernier n'étant défini que si f est régulière).

Cependant, si f n'est pas régulière, il est possible que $\mathrm{Expr}/\sim_f$ soit fini, ce qui est complètement contraire à ce qui se passe en théorie des langages sur mots finis.

Exemple 2.3.15 *Soit* $f(u) = \min_{e \in \mathrm{Expr}} \{|e|, \exists n \in \mathbb{N}, u = e(n)\}$*. Alors* $\sim_f$ *n'a qu'une seule classe d'équivalence :* $f(C[e](n))$ *est toujours borné par* $|C[e]|$*, et donc toutes les expressions sont équivalentes. Il en résulte que* $\mathrm{Expr}/\sim_f$ *n'a qu'un seul élément, et ne peut donc pas contenir de semigroupe de stabilisation reconnaissant* f *: un semigroupe de stabilisation à un seul élément reconnait la fonction constante* 0 *si* $I = \emptyset$ *et* ∞ *sinon. Ceci nous donne une preuve que* f *n'est pas régulière.*

En revanche, la situation ici est similaire à celle des langages de mots infinis, où les objets algébriques généralisant les semigroupes sont les ω-semigroupes. En effet, si deux langages de mots infinis coïncident sur les mots ultimement périodiques, alors leurs congruences syntactiques ne permettent pas de les distinguer [Arn85]. Ainsi le langage constitué des mots ultimement périodiques ne définira qu'une seule classe d'équivalence, de manière analogue à l'exemple ci-dessus.

2.3.8 Détails techniques sur les $\omega\sharp$-expressions

On donne ici quelques propriétés permettant de mieux comprendre comment se comportent les $\omega\sharp$-expressions. La Proposition 2.3.17 permet de faire le lien avec la variante de $\sharp$-expression introduite précédemment dans [CKL10], et les Lemmes 2.3.16 et 2.3.18 décrivent certaines invariances respectées par les $\omega\sharp$-expressions.

Lemme 2.3.16 *Si* $E \equiv_f E'$*, alors pour tout contexte* $C_1[x]$*,* $C_1[E] \equiv_f C_1[E']$*.*

Démonstration Soit E, E' et $C_1[x]$ définis par l'énoncé du lemme. Soit $C[x]$ un contexte. on définit $C'[x] = C[C_1[x]]$. La définition de la relation $\equiv_f$ implique $C'[E] \rightleftharpoons_f C'[e']$. D'où $C[C_1[e]] \rightleftharpoons_f C[C_1[E']]$.

C'est vrai pour tout $C[x]$ donc $C_1[E] \equiv_f C_1[E']$.

□

Proposition 2.3.17 *La relation* $\equiv_f$ *ne change pas si on restreint les contextes à ne comporter qu'une occurrence de* x*, comme c'est le cas dans [CKL10].*

Démonstration Soit $\equiv'_f$ la relation d'équivalence limitée aux contexte ne contenant qu'une seule occurence de x. Il suffit de montrer $E \equiv'_f E' \implies E \equiv_f E'$, car la réciproque est triviale.

Supposons $E \equiv'_f E'$, et soit $C[x_1, x_2]$ un contexte avec deux occurences de x, étiquetées x_1 et x_2. Alors $C[E] = C[x_1 \leftarrow E, x_2 \leftarrow E] \rightleftharpoons_f C[x_1 \leftarrow E, x_2 \leftarrow E'] \rightleftharpoons_f C[x_1 \leftarrow E', x_2 \leftarrow E'] = C[e']$. La généralisation à un nombre arbitraire d'occurences de x est évidente, et on obtient donc $E \equiv_f E'$. □

Le lemme suivant sera utile dans certaines preuves pour des raisons techniques, mais c'est également une assertion intuitive qui permet de mieux comprendre les phénomènes à l'oeuvre dans les fonctions de coût en général.

Lemme 2.3.18 (Désynchronisation) *Soit f une fonction de coût régulière, et $e \in$ Expr, bien formée pour $\mathbf{S}_f$, contenant N opérateurs de stabilisation, numérotés $\sharp_1, \ldots, \sharp_N$. Pour tout $i \in \{1, \ldots, N\}$, soit σ_i une fonction $\mathbb{N} \to \mathbb{N}$ avec $\sigma_i(n) \to \infty$. Alors*

$$f(e[\sharp_i \leftarrow \sigma_i(n) \text{ pour tout } i]) \to \infty \Leftrightarrow f(e(n)) \to \infty.$$

En d'autres termes, on peut remplacer certains des exposants n par n'importe quelle fonction de n tendant vers l'infini. Ceci ne change pas la nature de la suite relativement à f.

Démonstration Le résultat est intuitif : puisqu'on travaille toujours modulo $\approx$, grandir à des vitesses différentes a un effet sur les fonctions de correction, mais pas sur le comportement qualitatif du type « être borné ou pas ».

On introduit une notation qui permettra de simplifier les explications. Si a_n et b_n sont deux expressions qui contiennent la variable libre n, et à valeur dans $\mathbb{N}$, on dira que $a_n \underset{n\to\infty}{\bowtie} b_n$ si $\limsup a_n = \infty \Leftrightarrow \limsup b_n = \infty$.

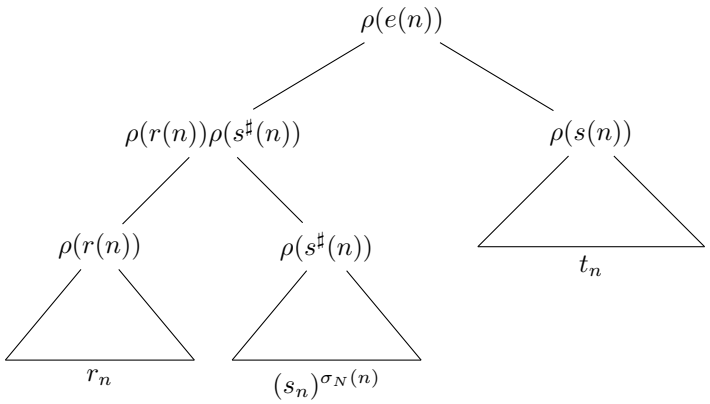
On peut remarquer qu'ici, toutes les suites seront soit bornées, soit tendant vers l'infini, grâce à la contrainte stipulant que e est bien formée pour $\mathbf{S}_f$. Par exemple e peut être obtenue à partir d'une $\omega\sharp$-expression, en remplaçant ω par $K_f!$.

On notera $e_n = e[\sharp_i \leftarrow \sigma_i(n)$ pour tout $i]$. On veut montrer que $f(e_n) \underset{n\to\infty}{\bowtie} f(e(n))$. Soit ρ compatible avec $\mathbf{S}_f$.

On va montrer qu'il existe α tel que pour tout n, $\rho(e_n) \sim_\alpha \rho(e(n))$, ce qui précise le résultat. On procède par induction sur N. Si $N = 0$, alors $e_n = e(n)$ donc le résultat est trivial.

On suppose le résultat vrai pour $k < N$ (avec la fonction $\alpha_<$), et on choisit $\sharp_N$ un opérateur de stabilisation extérieur (c'est-à-dire non imbriqué sous un autre $\sharp$). On peut écrire $e = r s^{\sharp_N} t$, avec $r, s, t \in$ Expr, bien formées pour $\mathbf{S}_f$, et $\mathrm{eval}(s) \in E(\mathbf{S}_f)$.

Par récurrence, il existe des n-calculs de hauteur bornée, et de valeur $\rho(r(n))$, $\rho(s(n))$ et $\rho(t(n))$ sur r_n, s_n et t_n respectivement. On peut ensuite combiner ces arbres au moyen de deux noeuds binaires et d'un noeud idempotent/stabilisant de la manière suivante :



L'arbre ainsi construit utilise parfois n comme seuil, parfois $\sigma_N(n)$. Plus précisément, c'est un n-sous-calcul et un $\sigma_N(n)$-sur-calcul. La suite de valeurs ainsi générée à la racine est donc $\sim$-équivalente à $\rho(e(n))$, tandis que le mot de feuilles est e_n. Ceci achève la preuve de $\rho(e_n) \sim \rho(e(n))$ □

2.4 Logiques de coût

2.4.1 La logique CFO

On étend la logique du premier ordre pour lui permettre de définir des fonctions de coût au lieu de langages. On appelle cette logique CFO (pour « Cost First-Order »), et elle est définie par la grammaire suivante :

$$\varphi := a(x) \mid x = y \mid x < y \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \forall x.\varphi \mid \forall^{\leq N} x.\varphi$$

où a est à valeur dans $\mathbb{A}$, et N est une unique variable libre à valeur dans $\mathbb{N}$.

Les variables sont à valeurs dans les positions du mot d'entrée, ordonnées par $<$. Cela signifie que si le mot d'entrée est de longueur n, les variables sont à valeurs dans $[0, n-1]$. Les opérateurs booléens $\wedge$ (ET) et $\vee$ (OU) ainsi que les quantificateurs $\exists, \forall$ ont la signification habituelle. Le nouveau prédicat $\forall^{\leq N} x.\varphi$ signifie que φ doit être vérifiée pour tout x, sauf pour au plus N « erreurs ».

On remarque que les négations, présentes dans FO ont été ici poussées aux feuilles. Ceci sert à garantir le fait que les prédicats $\forall^{\leq N} x.\varphi$ apparaissent toujours positivement dans les formules de CFO. On peut aussi autoriser les négations, mais en imposant cette contrainte de positivité pour les prédicats $\forall^{\leq N} x.\varphi$. On peut donc définir l'implication $\varphi \Rightarrow \psi$ à condition que φ ne contienne pas de prédicat $\forall^{\leq N} x$, en poussant aux feuilles les négations dans φ (chaque opérateur possède un dual excepté $\forall^{\leq N}$).

On peut définir les opérateurs $\neg a, \neq, \leq$ au moyen de ceux de la grammaire. Par exemple $\neg a(x) = \vee_{b \neq a} b(x)$.

Soit $u \in \mathbb{A}^+$, $n \in \mathbb{N}$, et φ une formule fermée. On dit que (u, n) est un modèle de φ (noté $(u, n) \models \varphi$) si φ est vraie sur la structure u, avec n comme valeur de

N (i.e. le nombre d'erreurs autorisé pour chaque opérateur $\forall^{\leq N}$). Si x est libre dans φ, on doit aussi lui donner une valeur, et on peut écrire $(u, n, i) \models \varphi$, où $i \in \mathbb{N}$ est la valeur donnée à x.

On peut maintenant associer une fonction de coût $[\![\varphi]\!] : \mathbb{A}^+ \to \mathbb{N}_\infty$ à toute formule close φ de CFO par :

$$[\![\varphi]\!](u) := \inf \{n \in \mathbb{N} : (u, n) \models \varphi\}.$$

On dit que $[\![\varphi]\!]$ est la fonction de coût *reconnue par* φ. Comme d'habitude, ces fonctions de coût sont considérées seulement modulo $\approx$.

Exemple 2.4.1 *Soit $a \in \mathbb{A}$ et $\varphi = \forall^{\leq N} x. \neg a(x)$. On a alors $[\![\varphi]\!](u) = |u|_a$.*

2.4.2 La logique CMSO

On définit CMSO (Pour « Cost Monadic Second-Order ») comme une extension de FO, où l'on peut aussi quantifier sur les ensembles. La syntaxe de CMSO est donc :

$$\begin{aligned}\varphi := \quad & a(x) \mid x = y \mid x < y \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x. \varphi \mid \forall x. \varphi \mid \forall^{\leq N} x. \varphi \mid \\ & \exists X. \varphi \mid \forall X. \varphi \mid x \in X \mid x \notin X.\end{aligned}$$

La sémantique des formules CMSO est définie de la même manière que pour FO : si $u \in \mathbb{A}^+$ et $n \in \mathbb{N}$, on dit que $(u, n) \models \varphi$ si φ est vraie sur la structure u, avec n comme valeur pour N. On définit alors $[\![\varphi]\!](u) := \inf \{n \in \mathbb{N} : (u, n) \models \varphi\}$.

Comme pour CFO, on peut définir l'implication $\varphi \Rightarrow \psi$ à condition que φ ne contienne pas de prédicat $\forall^{\leq N} x. \varphi$.

Exemple 2.4.2 *Soit $\mathbb{A} = \{a, b\}$. On définit la formule $bloc(X) = \forall x, y, z, (x < y < z \wedge x \in X \wedge z \in X) \Rightarrow y \in X)$ exprimant qu'un ensemble X est formé de positions consécutives et $a(X) = \forall x, x \in X \Rightarrow a(x)$, exprimant que X ne contient que des a.*

On définit maintenant $\varphi = \forall X, (bloc(X) \wedge a(X)) \Rightarrow (\forall^{\leq N} x, x \notin X)$.

Cette formule limite à N le nombre de a consécutifs dans le mot d'entrée. En conséquence, on a $[\![\varphi]\!] = \text{maxblock}_a$, définie dans la Section 1.3.

Cette logique a été introduite dans [Col11], d'une manière légèrement différente, comme une extension de MSO par des prédicats $|X| \leq N$ (où X est une variable du second ordre) apparaissant positivement dans la formule. Il n'est pas difficile de voir que ces deux formalismes sont équivalents.

Démonstration Soit CMSO' la logique MSO enrichi par le prédicat $|X| \leq N$, devant toujours apparaître positivement. On peut traduire toute formule de CMSO en une formule de CMSO en remplaçant toutes les occurences de « $\forall^{\leq N} x. \varphi(x)$ » par « $\exists X. (|X| \leq N \wedge \forall x. (x \in X \vee \varphi(x)))$ », où X est une variable du second ordre non utilisée dans le reste de la formule. Réciproquement, on peut aller de CMSO' à CMSO en remplaçant toutes les occurences de « $|X| \leq N$ » par « $\forall^{\leq N} x. x \notin X$ », où x est une variable du permier ordre non utilisée dans le

reste de la formule. Ces deux traductions préservent en fait la sémantique exacte, et pas seulement modulo $\approx$. □

Dans la suite, on pourra utiliser le prédicat « $|X| \leq N$ », défini comme « $\forall^{\leq N} x.x \notin X$ ».

Théorème 2.4.3 ([Col11]) *Les fonctions de coût régulières sont exactement celles reconnues par les formules de CMSO.*

Ce théorème indique la robustesse de la théorie des fonctions de coût : comme pour les langages, beaucoup de formalismes convergent vers la même notion de régularité.

2.4.3 Lien avec les langages

Remarque 2.4.4 *Toute formule de FO (resp. MSO) φ peut être vue comme une formule CFO (resp. CMSO). Dans ce cas, si L était le langage défini par φ, alors φ en tant que formule de coût reconnaît la fonction de coût $[\![\varphi]\!] = \chi_L$.*

Lemme 2.4.5 *Si φ est une formule de CFO (resp. CMSO) telle que $[\![\varphi]\!] \approx \chi_L$ pour un langage L, alors L est définissable par une formule de FO (resp. MSO).*

Démonstration $[\![\varphi]\!] \approx \chi_L$ signifie qu'il existe $n \in \mathbb{N}$ tel que pour tout $u \in L$, $[\![\varphi]\!](u) \leq n$, et pour tout $u \notin L$, $[\![\varphi]\!](u) = \infty$.

En particulier, tous les prédicats $\forall^{\leq N} x.\psi$ de φ sont satisfaits avec $N = n$, quand le mot est dans L. On peut donc remplacer chaque occurence de ces prédicats par $\exists x_1, x_2, \ldots, x_n. \forall x. (\psi \vee \bigvee_{i \in [1,n]} x = x_i)$, qui explicitent l'emplacement des n erreurs autorisées : $x_1, x_2, \ldots, x_n$. La formule résultante sera pure FO (resp. MSO), et reconnaîtra L. □

Corollaire 2.4.6 *CMSO est strictement plus expressive que CFO.*

Démonstration Il suffit de choisir un langage L MSO-définissable mais pas FO-définissable, comme $(aa)^*$. Par le Lemme 2.4.5, χ_L n'est pas CFO-définissable, mais par la Remarque 2.4.4 χ_L est CMSO-définissable. □

Chapitre 3

Fonctions temporelles

On va s'intéresser ici à une sous-classe des fonctions de coût régulières : les fonctions temporelles. L'intuition est de ne mesurer que des évènements consécutifs, ce qui correspond à des intervalles de temps dans l'exécution. L'étude de ces fonctions est l'objet du papier [CKL10], les résultats de cette section proviennent de cet article.

On peut signaler que plusieurs modèles quantitatifs d'automates dont l'objectif était de mesurer des évènements consécutifs ont été considérés dans la littérature [Kir04, KPV09]. La classe des fonctions de coût régulières permet de formaliser cette contrainte. Le modèle des *desert automata* [Kir04] correspond aux automates temporels décrits ici. La logique Prompt-LTL [KPV09] étend LTL en imposant une borne globale sur certains temps d'attente. Le lien précis entre la classe temporelle et Prompt-LTL sera décrit dans la Section 4.5.

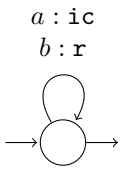
3.1 Définition par les automates

Un B-automate (resp. S-automate) avec compteurs Γ est *temporel* s'il utilise seulement les actions atomiques $\{\mathtt{ic}, \mathtt{r}\}^\Gamma$ (resp. $\{\mathtt{i}, \mathtt{cr}\}^\Gamma$).

C'est-à-dire que les automates temporels doivent actualiser leurs compteurs à chaque transition : ils ne peuvent pas faire de « pause » (action ε). Cela les force à ne compter que des évènements consécutifs dans le temps.

On dit qu'une fonction de coût est *B-temporelle* (resp. *S-temporelle*) si elle est reconnu par un B- (resp. S-) automate temporel. On verra plus tard que ces deux notions coïncident, toujours modulo $\approx$ (Théorème 3.2.9).

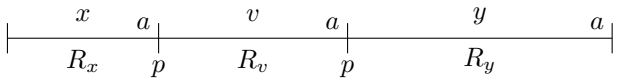
Exemple 3.1.1 *Sur l'alphabet $\mathbb{A} = \{a, b\}$, la fonction* maxblock_a *de la Section 1.3 est B-temporelle, comme le montre l'automate suivant :*



En revanche, la fonction $f_a : u \mapsto |u|_a$ n'est pas B-temporelle.

En effet, supposons que f_a est calculée par un B-automate temporel $\mathcal{A} = \langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta \rangle$ Soit g la fonction exacte calculée par $\mathcal{A}$, et soit α tel que $f_a \approx_\alpha g$. Soient $K = |Q| + 1$ et $N = \alpha(K) + 1$. Soit R une exécution de $\mathcal{A}$ sur $u = (b^N a)^K$ de valeur $g(u)$.

Considérons les états de R situés juste avant la lecture de chaque a dans u. Ils sont au nombre de K, mais $K > |Q|$, donc au moins deux d'entre eux doivent correspondre au même état p de l'automate. Ceci nous donne une décomposition de u en xvy avec $|v|_a \geq 1$, $|v| \geq N$, et l'exécution R est dans le même état p après x et xv.



Soient R_x, R_v, R_y les portions d'exécutions correspondantes. Supposons qu'il existe un compteur de $\mathcal{A}$ qui n'est pas remis à zéro durant R_v, alors il est incrémenté durant toute cette portion, et on obtient $g(u) \geq N > \alpha(|u|_a)$. Ceci contredit $g \approx_\alpha |\cdot|_a$. Tous les compteurs doivent donc être remis à zéro durant R_v. On considère maintenant pour tout $m \in \mathbb{N}$ le mot $u_m = xv^m y$. On peut vérifier que $|u_m|_a \geq m$, car $|v|_a \geq 1$. Cependant, pour tout $m \in \mathbb{N}$, l'exécution $R_x(R_v)^m R_y$ est un témoin de $g(u_m) \leq g(u) + |v|$, et donc $\{g(u_m), m \in \mathbb{N}\}$ est borné. Or, $\{f_a(u_m), m \in \mathbb{N}\}$ n'est pas borné, ce qui est absurde car $f_a \approx g$. On obtient donc la contradiction désirée.

3.2 Métronomes et langages uniformes

Dans cette section, on donne une caractérisation des fonctions B-temporelles et S-temporelles. Cette caractérisation a la particularité de revenir aux langages, et en particulier à une certaine classe de langages, qu'on appellera « langages métronomes ».

3.2.1 Définitions

L'idée intuitive qui sous-tend ce chapitre est la mesure du temps au moyen d'un enrichissement du mot d'entrée par une information unaire. Cette idée est déjà présente dans [KPV09], sous le nom « alternating-color technique ». On formalise ici cette notion de manière plus poussée, en établissant notamment le lien avec les langages réguliers.

Définition 3.2.1 *Un* métronome *c est un mot sur l'alphabet $\{\sqcup, \downarrow\}$. Le nom provient du fait qu'un tel objet peut modéliser les* battements *d'un métronome : la lettre $\sqcup$ décrit un instant de silence, et la lettre $\downarrow$ décrit le son produit par l'aiguille lors d'un battement. Un métronome factorise naturellement le temps en intervalles :*

$$c = (\sqcup^{n_1-1}\downarrow)(\sqcup^{n_2-1}\downarrow)\dots(\sqcup^{n_k-1}\downarrow)\sqcup^{m-1}.$$

On utilise cette factorisation pour définir les fonctions :

$$\max{-}\mathrm{seg}(c) = max\{n_1, \dots, n_k, m\} \in \mathbb{N}$$
$$\min{-}\mathrm{seg}(c) \mapsto \inf\{n_1, \dots, n_k\} \in \mathbb{N}_\infty$$

Remarquons l'asymétrie entre ces deux fonctions, relative au dernier intervalle.

Un métronome c est périodique de période *P si $n_1 = n_2 = \dots = n_k = P$, et $m \leq P$.*

On peut remarquer qu'un métronome est périodique de période P si et seulement si $\max{-}\mathrm{seg}(c) \leq P \leq \min{-}\mathrm{seg}(c)$. De plus, étant donnés n et P, il existe un unique métronome de longueur n et de période P. On peut remarquer également que $\max{-}\mathrm{seg}(c) = val_B(h_B(c)) + 1$, où h_B envoie $\sqcup$ sur `ic` et $\downarrow$ sur `r`. De la même façon, $\min{-}\mathrm{seg}(c) = val_S(h_S(c)) + 1$ où h_S envoie $\sqcup$ sur `i` et $\downarrow$ sur `cr`.

Si $u \in \mathbb{A}^+$, un métronome sur u est un métronome c de longueur $|u|$. Dans ce cas, on note $\langle u, c\rangle$ le mot sur l'alphabet $\mathbb{A} \times \{\sqcup, \downarrow\}$, obtenu en rassemblant u et c. On notera $\mathbb{M}$ (pour « métronome ») l'alphabet $\mathbb{A} \times \{\sqcup, \downarrow\}$.

Pour L un langage sur $\mathbb{M}$, on définit les fonctions suivantes, de $\mathbb{A}^+$ vers $\mathbb{N}_\infty$:

$$\langle\!\langle L\rangle\!\rangle_B : u \mapsto \inf\{\max{-}\mathrm{seg}(c) \ : \ c \text{ métronome sur } u,\ \langle u, c\rangle \in L\}$$
$$\langle\!\langle L\rangle\!\rangle_S : u \mapsto \sup\{\min{-}\mathrm{seg}(c) \ : \ c \text{ métronome sur } u,\ \langle u, c\rangle \notin L\} + 1$$

Lemme 3.2.2 *Pour tout $L \subseteq \mathbb{M}^+$, $\langle\!\langle L\rangle\!\rangle_B \leq \langle\!\langle L\rangle\!\rangle_S$.*

Démonstration Soit $u \in \mathbb{M}^+$, on considère le plus petit P tel que le métronome c de période P sur u soit tel que $\langle u, c\rangle \in L$. Si une telle période P n'existe pas, cela signifie que $\langle u, \sqcup^{|u|}\rangle \notin L$, et $\langle\!\langle L\rangle\!\rangle_S(u) = \infty$, le résultat est donc vrai dans ce cas.. Sinon, on a $\langle\!\langle L\rangle\!\rangle_B(u) \leq P$, et de plus, $\langle u, c'\rangle \notin L$, où c' est le métronome de période $P - 1$ sur u. D'où $\langle\!\langle L\rangle\!\rangle_B(u) \leq P \leq \langle\!\langle L\rangle\!\rangle_S(u)$. □

On montre que les sémantiques $\langle\!\langle\cdot\rangle\!\rangle_B$ et $\langle\!\langle\cdot\rangle\!\rangle_S$ peuvent se convertir simplement en automates temporels :

Fait 3.2.3 *Si L est régulier et L (resp. $\overline{L}$) est accepté par un automate $\mathcal{A}$ à n états, alors la fonction $\langle\!\langle L\rangle\!\rangle_B$ (resp. $\langle\!\langle L\rangle\!\rangle_S$) est reconnu de manière exacte par un B- (resp. S-) automate temporel $\mathcal{B}$ à n états et un compteur.*

Démonstration On a vu que $\max{-}\mathrm{seg} = val_B \circ h_B + 1$. Il suffit donc de remplacer dans $\mathcal{A}$ chaque transition de la forme $(p,(a,c),q)$ par une transition $(p,a,h_B(c),q)$ pour obtenir $\mathcal{B}$. La construction pour obtenir un S-automate est similaire, en partant d'un automate pour le complément de L. □

La définition suivante va permettre de caractériser les fonctions temporelles, on précisant les propriétés désirables sur de tels langages :

Définition 3.2.4 *Un langage* α-uniforme *(ou simplement* uniforme *s'il existe un tel* α*) est un langage* $L \subseteq \mathbb{M}^+$ *tel que* $\langle\!\langle L \rangle\!\rangle_B \approx_\alpha \langle\!\langle L \rangle\!\rangle_S$*. Une fonction* f est α-uniforme *s'il existe* L α*-uniforme tel que* $\langle\!\langle L \rangle\!\rangle_S \leq f \preccurlyeq_\alpha \langle\!\langle L \rangle\!\rangle_B$*. Dans ce cas on dit que* L *est* α-uniforme pour f*. Une fonction de coût est uniforme si elle est* $\approx$*-équivalente à une fonction* α*-uniforme pour un certain* α*. On note* $\mathcal{FU}$ *l'ensemble des fonctions de coût uniformes.*

On peut remarquer qu'il est suffisant de montrer que $\langle\!\langle L \rangle\!\rangle_S \preccurlyeq_\alpha \langle\!\langle L \rangle\!\rangle_B$ pour montrer que L est α-uniforme : le Lemme 3.2.2 donne l'autre direction dans tous les cas.

Exemple 3.2.5 *Soit* $L \subseteq \mathbb{A}^+$*, on pose* $K = \left\{ \langle u, c\rangle : c \in L, v \in \{\sqcup, \downarrow\}^{|u|} \right\}$*. Alors* K *est id-uniforme, et témoigne du fait que* χ_L *est id-uniforme.*

Exemple 3.2.6 *On reprend la fonction* $\mathrm{maxblock}_a$*, calculant la taille du plus gros bloc de* a*. L'idée « naturelle » serait de choisir le langage* $M = ((a,\sqcup) + (b,\downarrow))^+$*, et obtenir l'équivalence* $\langle\!\langle M \rangle\!\rangle_B \approx \mathrm{maxblock}_a$*. Cependant, un tel langage* M *n'est pas uniforme. Pour s'en rendre compte, considérons les mots* ba^m*, avec* $m \in \mathbb{N}$*. On a* $\mathrm{maxblock}_a(ba^m) = m$*, mais* $\langle\!\langle M \rangle\!\rangle_S(ba^m) = 0$*. Ceci contredit* $\langle\!\langle M \rangle\!\rangle_S \approx \mathrm{maxblock}_a$*.*

Cela provient du fait que le métronome témoignant de $\langle\!\langle M \rangle\!\rangle_B \approx \mathrm{maxblock}_a$ *dépend du mot d'entrée : il attend sur les* a *et bat sur les* b*. C'est en contradiction avec l'intuition importante selon laquelle le métronome mesure le temps, et devrait pouvoir être choisi indépendant du mot d'entrée.*

Il est possible de construire un langage uniforme L *pour* $\mathrm{maxblock}_a$*. Ce langage vérifie que chaque bloc de* a *contient au plus un battement du métronome :*

$$L = K[((b,\sqcup) + (b,\downarrow))K]^* \qquad \text{où} \quad K = (a,\sqcup)^* + (a,\sqcup)^*(a,\downarrow)(a,\sqcup)^* \ .$$

Soit $u \in \mathbb{A}^+$*, et* c *un métronome tel que* $\min{-}\mathrm{seg}(c) = n$ *et* $\langle u, c\rangle \notin L$*. Il existe donc un facteur de* u *de la forme* a^k *contenant deux battements de* c*. D'où* $k \geq n+1$*, et finalement* $\langle\!\langle L \rangle\!\rangle_S \leq \mathrm{maxblock}_a$*.*

Réciproquement, soit $u \in \mathbb{A}^+$ *et* c *un métronome tel que* $\max{-}\mathrm{seg}(c) = n$ *et* $\langle u, c\rangle \in L$*. Alors* $k = \mathrm{maxblock}_a(u)$*, et* u *contient un facteur* a^k*. Puisque* $\langle u, c\rangle \in L$*, ce facteur contient au plus un battement. D'où* $k \leq 2n-1$*. On obtient donc* $\mathrm{maxblock}_a < 2\langle\!\langle L \rangle\!\rangle_B$*. Finalement,* L *est* α*-uniforme pour* $\mathrm{maxblock}_a$*, avec* $\alpha : n \mapsto 2n$*.*

3.2.2 Propriétés de clôture

Exprimer une fonction de coût via un langage uniforme permet de pouvoir utiliser des résultats classiques de théories des langages.

Théorème 3.2.7 *Soit $L, M \subseteq \mathbb{M}^+$ des langages α-uniformes, $\mathbb{B}$ un alphabet fini, $h : \mathbb{A} \to \mathbb{B}$ et $g : \mathbb{B} \to \mathbb{A}$, des fonctions, étendues canoniquements aux mots. On a :*

- *$L \cup M$ est α-uniforme et $\langle\!\langle L \cup M \rangle\!\rangle_B = \min(\langle\!\langle L \rangle\!\rangle_B, \langle\!\langle M \rangle\!\rangle_B)$,*
- *$L \cap M$ est α-uniforme et $\langle\!\langle L \cap M \rangle\!\rangle_S = \max(\langle\!\langle L \rangle\!\rangle_S, \langle\!\langle M \rangle\!\rangle_S)$,*
- *$L_{\circ g} = \{\langle u, c\rangle : \langle g(u), c\rangle \in L\}$ est α-uniforme et $\langle\!\langle L_{\circ g} \rangle\!\rangle_B = \langle\!\langle L \rangle\!\rangle_B \circ g$,*
- *$L_{\inf,h} = \{\langle h(u), c\rangle : \langle u, c\rangle \in L\}$ est α-uniforme et $\langle\!\langle L_{\inf,h} \rangle\!\rangle_B = (\langle\!\langle L \rangle\!\rangle_B)_{\inf,h}$,*
- *$L_{\sup,h} = \mathbb{M}^+ \setminus \{\langle h(u), c\rangle : \langle u, c\rangle \notin L\}$ est α-uniforme et $\langle\!\langle L_{\sup,h} \rangle\!\rangle_S = (\langle\!\langle L \rangle\!\rangle_S)_{\sup,h}$*

Démonstration Les cinq cas suivent le même schéma de preuve. On traite ici le cas de l'inf-projection. On a

$$\begin{aligned}(\langle\!\langle L_{\inf,h} \rangle\!\rangle_B)(v) &= \inf\{\max{-}\mathrm{seg}(c) \ : \ \langle v, c\rangle \in L_{\inf,h}\} \\ &= \inf\{\max{-}\mathrm{seg}(c) \ : \ \langle u, c\rangle \in L, \ h(u) = v\} \\ &= \inf\{\inf\{\max{-}\mathrm{seg}(c) \ : \ \langle u, c\rangle \in L\} \ : \ h(u) = v\} \\ &= (\langle\!\langle L \rangle\!\rangle_B)_{\inf,h}(v)\end{aligned}$$

Il reste à montrer que si L est α-uniforme, alors $L_{\inf,h}$ l'est aussi. Soit v un mot sur $\mathbb{B}$ et c un métronome témoin de $\langle\!\langle L \rangle\!\rangle_B(v) = n$, i.e. tel que $\langle v, c\rangle \in L_{\inf,h}$ et $\max{-}\mathrm{seg}(c) = n$. Soit c' un métronome sur v tel que $\min{-}\mathrm{seg}(c') > \alpha(n)$, on doit montrer $\langle v, c'\rangle \in L_{\inf,h}$. Puisque $\langle v, c\rangle \in L_{\inf,h}$, il existe $u \in \mathbb{A}^+$ tel que $v = h(u)$ et $\langle u, c\rangle \in L$. Mais L est α-uniforme, donc $\langle u, c'\rangle \in L$. On peut conclure que $\langle v, c\rangle \in L_{\inf,h}$. □

Lemme 3.2.8 *Sur les alphabets $\{\mathtt{ic}, \mathtt{r}\}$ et $\{\mathtt{i}, \mathtt{cr}\}$, val_B et val_S sont $\times 2$-uniformes, avec $\times 2(n) = 2n$.*

Démonstration La preuve pour val_B est la même que dans l'Exemple 3.2.6, en remplaçant a par $\mathtt{ic}$ et b par $\mathtt{r}$.

On traite ici l'autre cas : on construit L régulier $\times 2$-uniforme pour val_S.

On dit qu'un métronome c est *compatible* avec un mot u sur $\{\mathtt{i}, \mathtt{cr}\}^+$ si u et c ont même longueur et si c bat au plus une fois dans tout bloc de $\mathtt{i}$ suivi d'un $\mathtt{cr}$ dans u. Dans la suite, on appellera *bloc* de u un bloc de $\mathtt{i}$ suivi d'un $\mathtt{cr}$ dans u. Soit $L = \{\langle u, c\rangle, c \text{ compatible avec } u\}$:

$$L = K[((\mathtt{cr}, \sqcup) + (\mathtt{cr}, \downarrow))K]^* K^* \text{ où } K = (\mathtt{i}, \sqcup)^* + (\mathtt{i}, \sqcup)^*(\mathtt{i}, \downarrow)(\mathtt{i}, \sqcup)^*.$$

On montre que L est $\times 2$-uniforme pour val_S. Il suffit de montrer que $val_S \preccurlyeq \langle\!\langle L \rangle\!\rangle_B$ et $\langle\!\langle L \rangle\!\rangle_S \preccurlyeq val_S$.

Soit $u \in \{a,b\}^+$, et c un métronome sur u tel que $\langle u, c\rangle \in L$ et $\max{-}\mathrm{seg}(c)$ est minimal. Soit $n = val_S(u)$ la taille du plus petit bloc de u. Si $\max{-}\mathrm{seg}(c) \leq \lfloor n/2 \rfloor$, alors il y a au moins deux $\downarrow$ de c dans le plus petit (donc dans tout) bloc de u, et donc c n'est pas compatible avec u. On a donc $\langle\!\langle L \rangle\!\rangle_B(u) = \max{-}\mathrm{seg}(c) > \lfloor n/2 \rfloor = \lfloor val_S(u)/2 \rfloor$. C'est vrai pour tout u, donc $val_S \preccurlyeq_{\times 2} \langle\!\langle L \rangle\!\rangle_B$.

On montre maintenant que $\langle\!\langle L \rangle\!\rangle_S \leq val_S$.

Soit $u \in \mathbb{A}^+$, et c un métronome sur u tel que $\langle u, c\rangle \notin L$ et $\min{-}\mathrm{seg}(c)$ est maximal. $\langle u, c\rangle \notin L$ donc il y a au moins deux $\downarrow$ de c dans tout bloc de u. Il s'ensuit $\min{-}\mathrm{seg}(c) \leq val_S(u)$. D'où par définition de c, $\langle\!\langle L \rangle\!\rangle_S(u) = \min{-}\mathrm{seg}(c) \leq val_S(u)$. C'est vrai pour tout u donc $\langle\!\langle L \rangle\!\rangle_S \leq val_S$. □

3.2.3 Lien avec les fonctions temporelles

Finalement, le théorème suivant établit une correspondance claire entre langages uniformes et fonctions de coût temporelles.

Théorème 3.2.9 *Si f est une fonction de coût régulière, les assertions suivantes sont équivalentes :*

1. *f est uniforme,*
2. *f est B-temporelle,*
3. *f est reconnue par un B-automate temporel à un compteur,*
4. *f est S-temporelle,*
5. *f est reconnue par un S-automate temporel à un compteur,*

Démonstration (1)⇒(3) et (1)⇒(5) sont conséquences du Fait 3.2.3.

(3)⇒(2) et (5)⇒(4) sont triviaux.

(2)⇒(1) :

Soit $\mathcal{A} = \langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta\rangle$ un B-automate temporel utilisant les compteurs $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$. Une exécution de $\mathcal{A}$ est un mot sur l'alphabet $\mathbb{B} = Q \times \mathbb{A} \times \{\mathtt{ic}, \mathtt{r}\}^\Gamma \times Q$. D'après la définition de $[\![\cdot]\!]_B$, pour tout $u \in \mathbb{A}^*$:

$$[\![\mathcal{A}]\!]_B(u) = \inf_{R \in \mathbb{B}^*} \{\max(\chi_K(R), val_B \circ \pi_1(R), \cdots, val_B \circ \pi_k(R)) \ : \ \pi_{\mathbb{A}}(R) = u\}$$

où $K \subseteq \Delta^*$ est l'ensemble (régulier) des exécutions valides de $\mathcal{A}$; pour tout $i \in [\![1, k]\!]$, π_i projette chaque transition (p, a, ν, q) sur la composante γ_i de ν (et est étendue aux mots canoniquement). Finalement $\pi_{\mathbb{A}}$ projette chaque transition (p, a, ν, q) sur a (également étendue aux mots). D'après l'Exemple 3.2.5, $\chi_K \in \mathcal{FU}$. D'après le Lemme 3.2.8, $val_B \in \mathcal{FU}$, et par le Théorème 3.2.7, $\mathcal{FU}$ est stable par composition, max et inf-projection. D'où $[\![\mathcal{A}]\!]_B \in \mathcal{FU}$.

(4)⇒(1) : la preuve est identique à celle ci-dessus, la seule différence est que la fonction χ_K est remplacée par $\chi_{\overline{K}}$. □

Plus précisément, le Théorème 3.2.7 et le Lemme 3.2.8 permettent d'affirmer que si une fonction f est donnée par l'une des 5 descriptions du Théorème 3.2.9,

alors on peut la décrire de n'importe laquelle des quatre autres par une fonction g telle que $f \approx_{\times 2} g$.

Dans la suite, on dira simplement qu'une fonction de coût f est *temporelle* au lieu de B-temporelle ou S-temporelle.

3.2.4 Discussion

On peut remarquer que les langages uniformes permettent de présenter une fonction de coût comme définie simultanément par un infimum ($\langle\!\langle \cdot \rangle\!\rangle_B$) ou un supremum ($\langle\!\langle \cdot \rangle\!\rangle_S$). Cette présentation « auto-duale » autorise à voir une telle fonction comme un B- ou un S-automate selon les besoins, ce qui est assez différent des formalismes introduits jusqu'ici.

Une autre nouveauté intéressante est la réduction de la descriptions de cette classe de fonctions de coût à la théorie des langages, via les langages uniformes. Ces langages peuvent profiter de tous les résultats classiques, et peuvent être manipulés sous n'importe quelle forme. On peut donc imaginer des optimisations des constructions sur les fonctions de coût temporelles, qui résulteraient de cette réduction.

Il faut cependant garder à l'esprit que deux langages distincts L, L' peuvent être tels que $\langle\!\langle L \rangle\!\rangle_B \approx \langle\!\langle L' \rangle\!\rangle_B$ (et même $\langle\!\langle L \rangle\!\rangle_B = \langle\!\langle L' \rangle\!\rangle_B$). Certaines procédures qui étaient optimales dans la théorie des langages peuvent perdre ce caractère sur des langages utilisés pour décrire des fonctions de coût temporelles, car il faut aussi choisir une bonne représentation. Par exemple la minimisation d'un automate (ou d'un semigroupe) pour L ne permet a priori pas d'obtenir un objet minimal pour $\langle\!\langle L \rangle\!\rangle_B$.

On peut également remarquer que les fonctions de coût temporelles présentent des caractéristiques beaucoup plus simples que dans le cas général : la fonction de correction $\times 2$ est la seule imprécision commise dans toutes les traductions (dans le cas général il faut une fonction polynomiale), et un seul compteur sur les automates suffit pour capturer toute la classe des fonctions temporelles.

Ces résultats montrent que cette classe pourrait être sujette à des applications pratiques, car elle correspond à une idée naturelle (mesurer le temps), et la complexité des objets sur cette classe est diminuée, tandis que la précision est améliorée.

La notion qui sous-tend celle de langage uniforme est l'idée que l'on peut mesurer une quantité en ajoutant une information unaire à l'environnement (les battements du métronome). Puisque cette information unaire peut être lue par un automate classique, celui-ci peut avoir accès à la quantité que l'on veut mesurer. Cette approche n'est pas forcément limitée au cas des fonctions temporelles sur les mots finis, et pourrait être étendue à de nombreux autres cas.

3.3 Caractérisation algébrique

On considère une classe de semigroupes de stabilisation, qui servira à caractériser les fonctions de coût temporelles :

Définition 3.3.1 *Soit* $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ *un semigroupe de stabilisation.* $\mathbf{S}$ *est un* semigroupe temporel *s'il ne comporte pas deux* $\mathcal{J}$*-classes régulières* $J \geq_{\mathcal{J}} J'$ *avec* J *stable et* J' *instable.*

On peut remarquer que cette définition est équivalente à la propriété suivante : pour tout élément stable a de $\mathbf{S}$, pour tout $x, y \in S$, si $x \cdot a \cdot y$ est régulier alors il est stable.

L'intuition derrière cette définition est la suivante : une $\mathcal{J}$-classe J est stable si elle est l'image de mots dont la quantité ne nous importe pas : la fonction de coût reconnue par $\mathbf{S}$ ne peut pas faire la différence entre « un peu » et « beaucoup » d'éléments de J. A contrario, une classe instable représente des éléments dont on veut mesurer le nombre d'occurences, car la fonction se comporte différemment suivant qu'il y en a un peu ou beaucoup ($a^\sharp \neq a$).

Cependant, si on avait un élement $x \cdot a \cdot y$ instable alors que a est stable, cela voudrait dire qu'on veut compter le nombre d'occurence de $x \cdot a \cdot y$ tout en faisant une « pause » en lisant a. C'est donc ce que l'on veut interdire si l'on se restreint à mesurer des évènements consécutifs.

Théorème 3.3.2 *Soit* f *une fonction de coût régulière, les assertions suivantes sont équivalentes :*

(1) f *est temporelle*

(2) f *est reconnue par un semigroupe temporel*

(3) $\mathbf{S}_f$*, le semigroupe de stabilisation minimal de* f*, est temporel*

Démonstration

(1) $\Rightarrow$ (2) Soit $\mathcal{A} = \langle Q, \mathbb{A}, In, Fin, \gamma, \Delta \rangle$ un B-automate temporel (on peut le prendre à un seul compteur d'après le Théorème 3.2.9), et f la fonction de coût reconnue par $\mathcal{A}$.

Le but est de lui associer un triplet $\mathbf{S}_\mathcal{A}, h, I$ qui reconnaît f et tel que $\mathbf{S}_\mathcal{A}$ est temporel.

On commence par construire un semigroupe temporel qui décrit la sémantique des actions du compteur de $\mathcal{A}$.

Soit $\mathbf{S}_\gamma = \langle S_\gamma, \cdot, \leq, \sharp \rangle$ avec $S_\gamma = \{\mathtt{ic}, \mathtt{r}, \bot\}$.

On pose $\bot \leq ic \leq r$, et les opérations $\cdot$ et $\sharp$ sont représentées par le tableau suivant :

FIGURE 3.1 – Produit d'actions

$\cdot$	r	ic	$\bot$	$\cdot^\sharp$
r	r	r	$\bot$	r
ic	r	ic	$\bot$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

On construit maintenant le semigroupe de stabilisation $\mathbf{S}_\mathcal{A}$ de la manière suivante :

Soit $S = S^{Q \times Q}_{\gamma}$.
Si $E, F \in S$, on définit :

$$\forall p, t \in Q, (E \cdot F)(p,t) = \max\{E(p,q) \cdot F(q,t) \ : \ q \in Q\},$$

et l'ordre par $E \leq F$ si pour tous $p, q \in Q, E(p,q) \leq F(p,q)$.

Finalement, pour E idempotent, on définit $E^\sharp$ par :

$$\forall p, q \in Q, E^\sharp(p,q) = \max\{E(p,t) \cdot E(t,t)^\sharp \cdot E(t,q) \ : \ t \in Q\}.$$

On définit maintenant $S_{\mathcal{A}} \subseteq S$ de la manière suivante. Soit $h : \mathbb{A} \to S$ définie par $h(a)(p,q) = \max\{\sigma / (p,a,\sigma,q) \in \Delta\}$ pour tout $a \in \mathbb{A}$ et $p, q \in Q$. Soit $S_{\mathcal{A}} = \langle h(\mathbb{A}) \rangle^\sharp$ l'ensemble de tous les éléments pouvant être obtenus à partir des éléments de $h(\mathbb{A})$ par produit et stabililsation.

Théorème 3.3.3 ([Col09]) $\mathbf{S}_{\mathcal{A}} = \langle S_{\mathcal{A}}, \cdot, \leq, \sharp \rangle$ *est un semigroupe de stabilisation, et en prenant* $I = \{E \in S_{\mathcal{A}} : \forall (p,q) \in In \times Fin, E(p,q) = \bot\}$, $\mathbf{S}_{\mathcal{A}}, h, I$ *reconnaît* $[\![\mathcal{A}]\!]_B$.

Il nous reste à montrer que $\mathbf{S}_{\mathcal{A}}$ est temporel.

Lemme 3.3.4 *Soit* E *idempotent et* $p, q \in Q$, *alors il existe* $t \in Q$ *tel que* $E(p,q) \leq E(p,t) \cdot E(t,t) \cdot E(t,q)$.

Démonstration Soit $k = |Q| + 1$. Puisque E est idempotent, on peut écrire $E = E \cdots E = E^k$, produit de longueur k. Par définition du produit dans $S_{\mathcal{A}}$, $E^k(p,q) = \max\{E(p,p_1) \cdot E(p_1,p_2) \ldots E(p_{k-1},q), p_1, \ldots, p_{k-1} \in Q\}$.

Soit $p_1, \ldots, p_{k-1}$ la suite d'états réalisant le maximum ci-dessus. On pose $p_0 = p$ et $p_k = q$. Pour tout $i \in [1,k]$, soit $a_i = E(p_{i-1}, p_i)$. Alors $E(p,q) = a_1 \cdots a_k$. Mais $k > |Q|$ donc il existe des entiers j, l tels que $1 < j < l < k$ et $p_j = p_l = t$. De plus, en réutilisant l'idempotence de E et la définition du produit via un maximum, on obtient $E(p,t) \geq a_1 \cdots a_j$, $E(t,t) \geq a_{j+1} \cdots a_l$ et $E(t,q) \geq a_{l+1} \cdots a_k$. On en conclut $E(p,q) \leq E(p,t) \cdot E(t,t) \cdot E(t,q)$. □

Lemme 3.3.5 *Soit* J *une* $\mathcal{J}$*-classe de* $\mathbf{S}_{\mathcal{A}}$, *alors* J *est instable si et seulement s'il existe* $E \in J$ *idempotent et* $p, q \in Q$ *tels que* $E(p,q) = \mathtt{ic}$.

Démonstration Soit J une $\mathcal{J}$-classe instable, il existe $E \in J$ idempotent avec $E^\sharp \neq E$. Mais $E^\sharp \leq E$, donc il existe $p, q \in Q$ tels que $E^\sharp(p,q) < E(p,q)$

D'après le Lemme 3.3.4, il existe $t \in Q$ tel que $E(p,q) \leq E(p,t) \cdot E(t,t) \cdot E(t,q)$. Mais $E^\sharp(p,q) \geq E(p,t) \cdot E(t,t)^\sharp \cdot E(t,q)$, donc $E(p,t) \cdot E(t,t) \cdot E(t,q) < E(p,t) \cdot E(t,t)^\sharp \cdot E(t,q)$. Ceci entraine $E(t,t)^\sharp \neq E(t,t)$, et finalement $E(t,t) = \mathtt{ic}$. On a montré l'implication $\Rightarrow$ du lemme. On a même obtenu en résultat un peu plus précis : on peut prendre $p = q$ dans l'énoncé du Lemme.

Réciproquement, soit $E \in J$ idempotent et $p, q \in Q$ tel que $E(p,q) = \mathtt{ic}$ Si on suppose J stable, on obtient $E^\sharp = E$ donc $E^\sharp(p,q) = \mathtt{ic}$. D'après la définition de $E^\sharp$, il existe $t \in Q$ tel que $ic = E(p,t) \cdot E(t,t)^\sharp \cdot E(t,q)$. D'après la définition du

produit dans S_γ, ceci implique $E(p,t) = E(t,t)^\sharp = E(t,q) = \mathtt{ic}$, et en particulier $E(t,t)^\sharp = \mathtt{ic}$, ce qui est impossible d'après la Figure 3.1.

J est donc instable, ce qui complète la preuve du lemme.

□

On combine maintenant ces lemmes pour montrer que $S_\mathcal{A}$ est temporel.

Soient J et J' deux $\mathcal{J}$-classes régulières de $\mathbf{S}_\mathcal{A}$ avec $J \geq_\mathcal{J} J'$ et J stable. D'après le Lemme 3.3.5, pour tout $E \in J$ idempotent, pour tous $s,t \in Q$, on a $E(s,t) \neq \mathtt{ic}$.
Soit $E' \in J'$, $J \geq_\mathcal{J} J'$ donc il existe $E \in J, A, B \in \mathbf{S}_\mathcal{A}, E' = A \cdot E \cdot B$. Supposons qu'il existe $p,q \in Q, E'(p,q) = \mathtt{ic}$. Dans ce cas il existe $s,t \in Q$ tel que $\mathtt{ic} = A(p,s) \cdot E(s,t) \cdot B(t,q)$. La table de porduit du semigroupe $\mathbf{S}_\gamma$ implique $E(s,t) = \mathtt{ic}$, ce qui est absurde. Notons que c'est seulement ici que l'on utilise l'hypothèse selon laquelle l'automata $\mathcal{A}$ est temporel. Avec un B-automate général, on aurait pu avoir $E(s,t) = \varepsilon$, et par exemple $A(p,s) = \mathtt{ic}$. E' ne peut pas contenir d'élément $\mathtt{ic}$, c'est donc un idempotent stable d'après le Lemme 3.3.5. J' est donc une classe stable. On a montré que $\mathbf{S}_\mathcal{A}$ est temporel, car si $J \geq_\mathcal{J} J'$ et J est stable, alors J' est stable.

(1) $\Rightarrow$ (2)

On veut maintenant associer un B-automate temporel à tout semigroupe temporel. Soit $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ un semigroupe temporel, et ρ une fonction compatible avec $\mathbf{S}$.

Définition 3.3.6 *On pose*

$$\mathit{Instab} = \{x \in \mathbf{S} : \exists e \in E(\mathbf{S}), e^\sharp \neq e \text{ and } e \leq_\mathcal{J} x\}.$$

Instab est donc une union de classes instables ou irrégulières, incluant toutes les classes instables. On définit Stab comme son complément, qui contiendra donc toutes les classes stables et possiblement des classes irrégulières.

Remarque 3.3.7 *Dans la partie stable,* $\mathbf{S}$ *est en fait un semigroupe classique, et admet donc le produit comme fonction compatible. On peut en conclure qu'il existe* η *tel que*

$$\forall u \in \mathit{Stab}^+,\ \rho(u) \sim_\eta \pi(u).$$

Soit f reconnue $\mathbf{S}, h, I$, avec I un idéal de $\mathbf{S}$ et $h : \mathbb{A} \to S$.

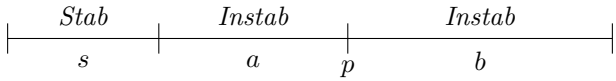
Soit α_ρ donné par le Théorème 2.2.14.

On construit un B-automate temporel $\mathcal{A} = \langle Q, \mathbb{A}, \mathit{In}, \mathit{Fin}, \{\gamma\}, \Delta \rangle$ qui reconnaît f. Si u est un mot lu par notre automate, on veut trouver les facteurs de u qui s'évaluent en un idempotent instable répété un grand nombre de fois. On pourra ainsi stabiliser cet élément dans $\mathbf{S}$.

L'idée est de deviner de tels facteurs de manière non déterministe, de compter leur longueur au moyen d'incréments, et de leur appliquer l'opérateur $\sharp$ quand ils deviennent trop longs.

On choisit $Q = (\{1\} \cup \mathit{Stab}) \times (\{1\} \cup \mathit{Instab}) \times (\{1\} \cup \mathit{Instab})$.

Ceci correspond à découper le mot lu jusqu'à présent de la manière suivante :



Où b est le facteur idempotent qui va se répéter. La seule position à deviner de manière non déterministe est p .

La table de transitions Δ de $\mathcal{A}$ est définie par l'ensemble suivant :

$$\{((s,a,1),l,ic,(s,a\cdot h(l),1)) : a\cdot h(l)\in \mathit{Instab}\} \quad (1)$$
$$\bigcup\{((s,a,b),l,ic,(s,a,b\cdot h(l))) : a\cdot b\cdot h(l)\in \mathit{Instab}\} \quad (2)$$
$$\bigcup \begin{array}{l}\{((s,a,b),l,r,(s\cdot a\cdot (b\cdot h(l))^\sharp,1,1)) :\\ a\cdot b\cdot h(l)\in \mathit{Instab}, b\cdot h(l) \text{ idempotent}\}\end{array} \quad (3)$$
$$\bigcup\{((s,a,b),l,r,(s\cdot a\cdot b\cdot h(l),1,1)) : a\cdot b\cdot h(l)\in \mathit{Stab}\} \quad (4)$$

On choisit $In = \{(1,1,1)\}$ (rien n'a encore été lu) et $Fin = \{(s,a,b), s\cdot a\cdot b\notin I\}$ (l'élément correspondant à l'évaluation du mot lu ne doit pas être dans I).

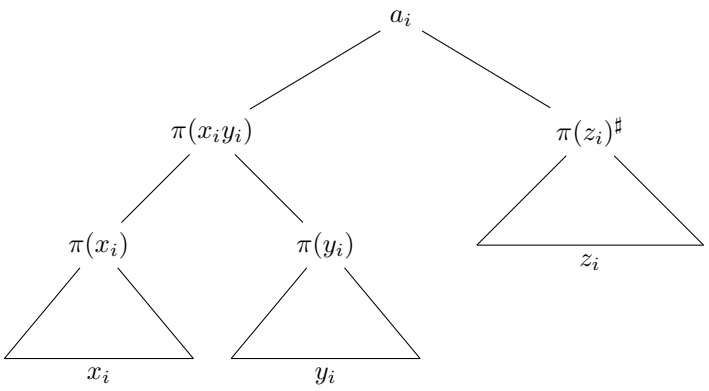
On commence par montrer que $f \preccurlyeq_\alpha [\![\mathcal{A}]\!]_B$, pour un certain α. Soit $u\in\mathbb{A}^+$, et R une exécution valide de $\mathcal{A}$ sur u terminant dans l'état (s,a,b). Soit $n = val_B(R)$, $q = s\cdot a\cdot b\notin I$, et $w = h(u)\in S^+$. Par définition de $\mathcal{A}$, w peut être factorisé en $x_1y_1z_1\dots x_ky_kz_k$ avec $q = \pi(x_1y_1)\cdot\pi(z_1)^\sharp\cdots\pi(x_ky_k)\cdot\pi(z_k)^\sharp$, et pour tout $j\in[1,k], \pi(x_j)\in \mathit{Stab}\cup\{1\}, \pi(y_j)\in \mathit{Instab}\cup\{1\}$, et $\pi(z_j)$ idempotent instable avec $|y_jz_j|\leq n$ (on suppose sans perte de généralité que la dernière transition est de type (3)). Les transitions utilisées pour la lecture des x_j sont de types (1), (2) et (4) ; celles correspondant aux y_j sont de type (1), et finalement, celles des z_j sont de type (2) avec une de type (3) à la fin (qui stabilise le facteur).

Pour $i\in[1,k]$, soit $a_i = \pi(x_iy_i)\cdot\pi(z_i)^\sharp$. Alors $a_i\in \mathit{Stab}$. On peut donc construire un n-calcul t sur $a_1\dots a_k$ qui n'utilise pas de noeud de stabilisation, de profondeur au plus $H = 3|\mathit{Stab}|$ et de valeur q.

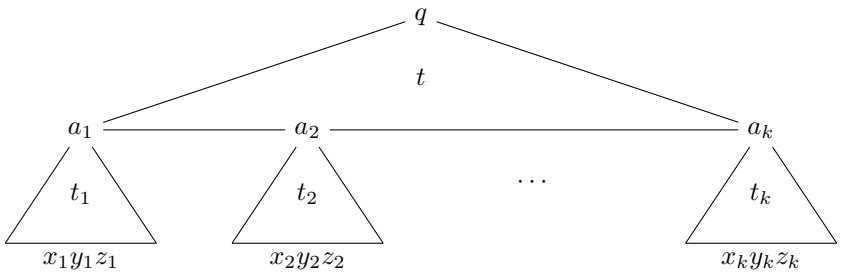
De même, pour tout $i\in[1,k]$ on peut construire un n-calcul sur $x_i\in\mathit{Stab}$ de valeur $\pi(x_i)$, et sur y_i de valeur v_i. Or, les y_i ont longueur au plus n, donc on n'aura pas de noeud de stabilisation dans le n-calcul sur y_i, d'où $v_i = \pi(y_i)$. En ajoutant un noeud binaire, on obtient un n-calcul sur x_iy_i de valeur $\pi(x_iy_i)$.

z_i est également de longueur inférieure à n, donc il existe un n-calcul de valeur $\pi(z_i)$ sur z_i. Cependant, on veut obtenir $a_i\in\mathit{Stab}$, donc on va mimer le comportement de l'automate, et construire en réalité un n-sous-calcul sur z_i, de valeur $\pi(z_i)^\sharp$.

On obtient ainsi le n-sous-calcul t_i suivant, sur $x_iy_iz_i$, de valeur a_i :



On construit finalement le n-sous-calcul t_u suivant sur u, en combinant t et les t_i :



Puisque $q \notin I$ (et par les Lemmes 2.2.13 et 2.2.23), ce n-sous-calcul est une preuve que $f(u) \leq_\alpha n$, où α ne dépend pas de u. C'est vrai pour tout R de valeur n, donc en passant à l'infimum on a bien $f(u) \preccurlyeq_\alpha [\![\mathcal{A}]\!](u)$. On peut conclure $f \preccurlyeq_\alpha [\![\mathcal{A}]\!]$

Réciproquement, étant donné un mot u et $n = f(u)$, on veut construire une exécution de $\mathcal{A}$ sur u témoignant $[\![\mathcal{A}]\!](u) \leq_\beta n$, pour un certain β (ne dépendant pas de u).

Lemme 3.3.8 *Soit $v \in \mathbb{A}^+$ un mot, et t un n-calcul de hauteur $d \geq 0$ sur $h(v)$. Soit x la valeur de t, c'est-à-dire l'étiquette de sa racine. Il existe une fonction de correction α_d ne dépendant que de d telle que pour tout état $(s_0, a_0, 1) \in Q$, il existe une exécution de $\mathcal{A}$ sur v partant de $(s_0, a_0, 1)$ de valeur au plus $\alpha_d(n)$, et d'état final $(s, a, 1)$ avec $s \cdot a = s_0 \cdot a_0 \cdot x$. Si de plus x est instable, on peut toujours choisir d'arriver dans l'état (s, a, x).*

Démonstration On montre le résultat par récurrence sur d. Si $d = 0$, l'exécution doit juste emprunter la transition (1) dans le premier cas et la tansition (4) dans le second, on peut prendre $\alpha_d(0) = 1$. Soit $d > 0$, on suppose le résultat vrai pour les hauteurs inférieures à d. On se fixe $(s_0, a_0, 1) \in Q$, et on veut construire une exécution respectant les propriétés voulues.

Il y a trois cas possibles pour la nature du noeud racine de t.

Premier cas : noeud binaire

Supposons que la racine de t est un noeud binaire reliant les sous-arbres t_1 et t_2, de feuilles $h(u_1)$ et $h(u_2)$, et de valeurs x_1 et x_2. On commence par utiliser l'hypothèse d'induction sur t_1 : il existe une exécution R_1 partant de l'état $(s_0, a_0, 1)$, de valeur au plus $\alpha_{d-1}(n)$, et arrivant dans un état $(s_1, a_1, 1)$ avec $s_1 a_1 = s_0 a_0 x_1$. En utilise ensuite de la même manière l'hypothèse d'induction sur t_2, de manière à construire une exécution sur u_2 partant cette fois de l'état $(s_1, a_1, 1)$. On arrive donc en $(s_2, a_2, 1)$ avec $s_2 a_2 = s_0 a_0 x_1 x_2$.

De plus, si $x = x_1 x_2$ est instable, cela signifie que x_1 et x_2 le sont aussi, et on peut donc choisir d'arriver dans l'état $(s_0, a_0, x_1 x_2)$ par hypothèse de récurrence.

En combinant R_1 et R_2, on construit une exécution R sur $u_1 u_2$, de valeur au plus $2\alpha_{d-1}(n)$, vérifiant les conditions voulues.

Second cas : noeud idempotent

Supposons que la racine de t est un noeud idempotent reliant les sous-arbres $t_1 \dots t_k$ avec $k \leq n$, de feuilles $h(u_1) \dots h(u_k)$, et tous de valeur $e \in E(\mathbf{S}_{\mathcal{A}})$. La valeur de t est alors e.

De la même manière que précédemment, en utilisant seulement des transitions de type (1) et (4), on peut construire des sous-exécutions de $\mathcal{A}$ sur chacun des t_i, chacune partant du point d'arrivée de la précédente, de valeurs inférieures à $\alpha_{d-1}(n)$, et effectuant le produit avec e dans l'état courant.

En combinant toutes ces exécutions, on construit une exécution R sur $u_1 \dots u_k$, de valeur au plus $n\alpha_{d-1}(n)$, vérifiant les conditions voulues.

Troisième cas : noeud de stabilisation

Supposons que la racine de t est un noeud de stabilisation reliant les sous-arbres $t_1 \cdot t_k$ avec $k > n$, de feuilles $h(u_1) \cdot h(u_k)$, et tous de valeur $e \in E(\mathbf{S}_{\mathcal{A}})$.

Si e est stable, on peut se contenter de faire le produit comme précédemment, en utilisant des transitions de type (1) et (4) seulement. Puisque chaque sous-arbre va induire un reset, l'exécution résultante aura valeur au plus $\alpha_{d-1}(n)$.

On suppose donc e instable dans la suite.

Par hypothèse de récurrence, il existe une exécution R_1 de $\mathcal{A}$ sur u_1, partant de $(s_0, a_0, 1)$ et arrivant en (s_0, a_0, e), de valeur au plus $\alpha_{d-1}(n)$. On change la dernière transition de cette exécution (de type (2)), de manière à utiliser plutôt une transition de type (3), et effectuer ainsi un reset. On arrive ainsi dans l'état $(s_0 a_0(e^\sharp), 1, 1)$. On peut repartir de cet état pour construire R_2 sur t_2, et de même manière terminer par une transition de type (3). Puisque $e^\sharp$ est idempotent, on revient dans le même état. Cette exécution a valeur au plus $\alpha_{d-1}(n)$. On réitère ce processus sur touts les t_i, pour $i \in [1, k]$. On construit ainsi une exécution de $\mathcal{A}$ sur $u_1 \dots u_k$, de valeur au plus $\alpha_{d-1}(n)$, arrivant dans l'état $(s_0 a_0(e^\sharp), 1, 1)$.

Dans tous les cas, on peut prendre $\alpha_d(n) = \max(n, 2)\alpha_{d-1}(n)$, ce qui conclut la preuve du Lemme 3.3.8. □

On peut donc terminer la preuve que $[\![\mathcal{A}]\!](u) \preccurlyeq_\beta f(u)$.

Par définition de $n = f(u)$, il existe une constante $H \in \mathbb{N}$ indépendante de u tel qu'il existe un n-calcul t de hauteur H sur u, et de valeur $q \notin I$. Par le Lemme

3.3.8, il existe une exécution R de $\mathcal{A}$ sur u de valeur au plus $\alpha_H(n)$, partant de l'état $(1,1,1)$, et arrivant dans un état (s,a,b) avec $s \cdot a \cdot b = q$. Puisque $q \notin I$, l'exécution R est acceptante, et est donc un témoin de $[\![\mathcal{A}]\!](u) \leq \alpha_H(n)$.

Ceci est vrai pour tout u, on a donc montré $[\![\mathcal{A}]\!](u) \preccurlyeq_\beta f(u)$, avec $\beta = \alpha_H$.

Ceci conclut la preuve de $[\![\mathcal{A}]\!] \approx f$.

□

Il reste à montrer (2) ⇔ (3) dans le Théorème 3.3.2. Pour ceci, il suffit de montrer que le caractère temporel est préservé par quotient :

Théorème 3.3.9 *Si* $\mathbf{S}$ *est temporel, et* $\equiv$ *est une congruence sur* $\mathbf{S}$*, alors* $\mathbf{S}/\equiv$ *l'est aussi.*

Démonstration On suppose que $\mathbf{S}/\equiv$ n'est pas temporel. Cela signifie qu'il existe $x, y \in E(\mathbf{S}/\equiv)$ avec $x^\sharp \neq x$, $y^\sharp = y$, et $x \leq_\mathcal{J} y$, c'est à dire qu'il existe $s, t \in \mathbf{S}/\equiv$ avec $x = syt$. Soit τ la projection canonique de $\mathbf{S}$ sur $\mathbf{S}/\equiv$, τ est un morphisme de semigroupes de stabilisation, et τ et surjectif. Il existe $b, c, d \in \mathbf{S}$ tel que $\tau(b) = y$, $\tau(c) = s$ et $\tau(d) = t$. Soit $a = cb^\sharp d$, alors $\tau(a) = \tau(c)\tau(b)^\sharp\tau(d) = sy^\sharp t = syt = x$, et par définition $a \leq_\mathcal{J} b^\sharp$.

On a $\tau(a^\sharp) = \tau(a)^\sharp = x^\sharp \neq x = \tau(a)$ donc $a^\sharp \not\equiv a$, ce qui implique que a est instable. Mais $a \leq_\mathcal{J} b^\sharp$, et $b^\sharp$ est stable, donc $\mathbf{S}$ n'est pas temporel. □

Théorème 3.3.10 *Etant donné une fonction de coût régulière* f*, on peut décider si* f *est temporelle.*

Démonstration Il suffit de calculer le semigroupe de stabilisation minimal $\mathbf{S}_f$ de f, et de tester si $\mathbf{S}_f$ est temporel. La relation $\mathcal{J}$ est calculable en tant polynomial, et il reste ensuite à vérifier que tous les couples d'idempotents vérifient la condition sur les semigroupes temporels, i.e. si $a \geq_\mathcal{J} b$ et a stable alors b stable. Si f est donnée par un automate et non par un semigroupe de stabilisation, on peut obtenir un semigroupe de stabilisation pour f par le Théorème 2.2.24, qui est effectif. □

La classe des fonctions de coût temporelles possède donc de nombreuses propriétés désirables. Elle témoigne de la robustesse de la technnique de Schützenberger, qui permet de relier la manière dont certaines contraintes se propagent à travers les différents modèles de reconnaissance. Il est également surprenant que bien que cette classe soit spécifique aux fonctions de coût (elle n'a pas d'analogue dans la théorie des langages), on peut la caractériser en termes de langages réguliers, au moyen des langages métronomes. L"étude de cette classe peut probablement mener à des applications pratiques, par exemple dans le domaine de la vérification, puisqu'il est naturel de vouloir borner le temps d'exécution de programmes, ou le temps d'attente maximal de réaction d'un système à une requête.

Chapitre 4

Fonctions de coût apériodiques

On va maintenant s'intéresser à une autre manière de spécifier des langages ou des fonctions de coût : la logique temporelle linéaire (LTL). Ce formalisme est l'un des plus utilisés en pratique, en particulier dans le domaine de la spécification. En effet, grâce à l'absence de variables, et à un choix d'opérateurs exprimant des propriétés naturelles, il est relativement facile de spécifier au moyen de LTL des conditions que l'on veut voir vérifiées par un système. D'un point de vue théorique, de nombreux résultats satisfaisant ont été obtenus en théorie des langages. Le problème consistant à vérifier la véracité d'une formule est PSPACE-complet. On dispose également du théorème suivant :

Théorème 4.0.11 ([Sch65, DG08]) *Pour les langages, il est équivalent d'être reconnu par :*
- *FO,*
- *semigroupes apériodiques,*
- *expressions rationnelles sans étoile (mais avec complémentation),*
- *LTL.*

Ceci permet notamment d'obtenir la décidabilité du problème de l'appartenance à la classe des langages LTL-définissables. Ce théorème permet également de mettre en lumière l'utilité des semigroupes dans le contexte des langages rationnels.

On va ici garder l'esprit de ce théorème, en le généralisant à la théorie des fonctions de coût. Les résultats de cette section proviennent de l'article [Kup11].

4.1 La logique CLTL

On étend la Logique Temporelle Linéaire (LTL), en incluant des notions quantitatives, pour lui permettre de définir des fonctions de coût et non plus des langages. Pour ceci, on ajoute un nouvel opérateur $\mathbf{U}^{\leq N}$, qui exprime une

contrainte quantitative. On note *CLTL* (pour « Cost LTL ») cette nouvelle logique.

4.1.1 Syntaxe

On commence par définir ici CLTL sur les mots finis, on utilise donc un opérator Ω qui permet de repérer la fin du mot. Ceci est inhabituel car dans la littérature, on considère plutôt LTL sur les mots infinis.

Les formules de CLTL sur un alphabet $\mathbb{A}$ sont définies par la grammaire suivante :

$$\varphi := a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid \ \varphi \mathbf{U}^{\leq N} \varphi \mid \Omega$$

Comme dans les logiques CFO et CMSO, N est une variable libre unique qui compte un nombre d'erreurs.

Les lettres $\mathbf{X}, \mathbf{U}$ siginifient respectivement « Next », « Until », et $\mathbf{U}^{\leq N}$ est une variante de cet opérateur qui autorise des erreurs. L'opérateur quantitatif $\mathbf{U}^{\leq N}$ doit apparaitre positivement dans la formule, ce qui explique l'absence de négation dans la syntaxe. Il serait également possible d'introduire des négations (notées $\neg$), en imposant que tous les opérateurs $\mathbf{U}^{\leq N}$ apparaissent sous un nombre pair de négations. On pourrait alors obtenir une formule de CLTL sans négations, en les poussant aux feuilles de la manière suivante, et en annulant les doubles négations :

- $\neg a = \bigvee_{b \neq a} b \vee \Omega$,
- $\neg(\varphi \wedge \psi) = (\neg\varphi) \vee (\neg\psi)$,
- $\neg(\varphi \vee \psi) = (\neg\varphi) \wedge (\neg\psi)$,
- $\neg\mathbf{X}\varphi = \Omega \vee \mathbf{X}(\neg\varphi)$,
- $\neg(\varphi \mathbf{U} \psi) = (\neg\psi)\mathbf{U}(\neg\varphi \vee \Omega)$.

Remarquons que l'on n'a pas besoin d'introduire dans la syntaxe un opérateur « Release » $\mathbf{R}$ pour nier l'opérateur $\mathbf{U}$, car la présence de l'opérateur Ω nous permet de définir $\psi \mathbf{R} \varphi = \varphi \mathbf{U}(\psi \vee \Omega))$. On utilisera la notation $\mathbf{R}$ dans la suite.

4.1.2 Sémantique

On veut associer une fonction de coût $[\![\varphi]\!] : \mathbb{A}^* \to \mathbb{N}$ à toute formule φ de CLTL.

Les formules de CLTL sont évaluées sur des mots, à partir d'une position précise, et avec une valeur pour N. On dira que (u, n, i) est un modèle de φ, noté $(u, n, i) \models \varphi$, si φ est vraie sur u à partir de la position i, avec n comme valeur pour N.

Soit $u = a_0 a_1 \dots a_k$, $i \in \mathbb{N}$ et $n \in \mathbb{N}$. On donne une sémantique à la syntaxe de CLTL, en définissant la valeur de vérité de $(u, n, i) \models \varphi$ par induction sur φ :

- $(u, n, i) \models a$ si $i \leq k$ et $a_i = a$;
- $(u, n, i) \models \Omega$ si $i = k + 1$;
- $(u, n, i) \models \varphi \wedge \psi$ si $(u, n, i) \models \varphi$ et $(u, n, i) \models \psi$;
- $(u, n, i) \models \varphi \vee \psi$ si $(u, n, i) \models \varphi$ ou $(u, n, i) \models \psi$;
- $(u, n, i) \models \mathbf{X}\varphi$ si $(u, n, i + 1) \models \varphi$;

– $(u,n,i) \models \varphi \mathbf{U} \psi$ s'il existe $j > i$ tel que $(u,n,j) \models \psi$ et pour tout $i \leq j' < j$, $(u,n,j') \models \varphi$;
– $(u,n,i) \models \varphi \mathbf{U}^{\leq N} \psi$ s'il existe $j > i$ tel que $(u,n,j) \models \psi$ et pour tout $i \leq j' < j$ sauf au plus n positions, $(u,n,j') \models \varphi$;

Le nouvel opérateur $\mathbf{U}^{\leq N}$ est donc une variante du Until $\mathbf{U}$, qui autorise au plus N erreurs.

Si la position d'évaluation i est omise, on utilisera la valeur par défaut 0, correspondant à la première lettre du mot. Ainsi, $(u,n) \models \varphi$ est une abréviation pour $(u,n,0) \models \varphi$.

On définit la fonction de coût reconnue par une formule φ, par

$$[\![\varphi]\!](u) = \inf \{n \in \mathbb{N} : (u,n) \models \varphi\}$$

Remarquons que si $(u,n) \models \varphi$, alors pour tout $k \geq n, (u,k) \models \varphi$, puisque les opérateurs $\mathbf{U}^{\leq N}$apparaissent toujours positivement. En particulier, $[\![\varphi]\!](u) = 0$ signifie que $\forall n \in \mathbb{N}, (u,n) \models \varphi$, et $[\![\varphi]\!](u) = \infty$ siginifie que $\forall n \in \mathbb{N}, u, n \not\models \varphi$ (puisque $\inf \emptyset = \infty$). Comme d'habitude, si φ est une formule n'utilisant pas d'opérateur $\mathbf{U}^{\leq N}$, c'est une formule LTL classique reconnaissant un langage L, et $[\![\varphi]\!] = \chi_L$.

On peut définir $\mathsf{true} = (\bigvee_{a \in \mathbb{A}} a) \vee \Omega$ and $\mathsf{false} = a \wedge \Omega$ (avec $a \in \mathbb{A}$ quelconque).

On enrichit également la syntaxe des opérateurs temporels par l'opérateur, « Futur » : $\mathbf{F}\varphi = \mathsf{true}\mathbf{U}\varphi$ (φ est vraie à un moment dans le futur) et « Globalement » : $G\varphi = \varphi \mathbf{U} \Omega$ (φ est toujours vraie à partir de la position courante), ainsi que sa version quantitative $\mathbf{G}^{\leq N}\varphi = \varphi \mathbf{U}^{\leq N} \Omega$, qui signifie que φ est vraie à toute position du mot à partir de la position courante, sauf pour au plus N erreurs.

Remarque 4.1.1 *On aurait pu utiliser la sémantique stricte pour le « Until » : $(u,n,i) \models \varphi \mathbf{U} \psi$ s'il existe $j > i$ tel que $(u,n,j) \models \psi$ et pour tout $i < j' < j$, $(u,n,j') \models \varphi$. Cela aurait permis d'économiser l'opérateur « Next » $\mathbf{X}$, en le définissant comme $\mathbf{X}\varphi = \mathsf{false}\mathbf{U}\varphi$.*

Exemple 4.1.2 *Pour tout $u \in \mathbb{A}^*$, $a \in \mathbb{A}$, et φ, ψ deux formules de CLTL, on a*

– $[\![a]\!](u) = 0$ *si* $u \in a\mathbb{A}^*$, *et* ∞ *sinon,*
– $[\![\Omega]\!](u) = 0$ *si* $u = \varepsilon$, *et* ∞ *sinon*
– $[\![\varphi \wedge \psi]\!] = \max([\![\varphi]\!], [\![\psi]\!])$, *et* $[\![\varphi \vee \psi]\!] = \min([\![\varphi]\!], [\![\psi]\!])$
– $[\![X\varphi]\!](au) = [\![\varphi]\!](u)$, $[\![X\varphi]\!](\varepsilon) = \infty$
– $[\![\mathsf{true}]\!] = 0$, *et* $[\![\mathsf{false}]\!] = \infty$
– $[\![\mathbf{G}^{\leq N}(\neg a)]\!](u) = |u|_a$

On utilisera les formules de CLTL pour décrire des fonctions de coût, donc les sémantiques $[\![\varphi]\!]$ sont toujours considérées modulo $\approx$.

4.2 De CLTL aux B-automates

On donne dans cette section une traduction directe des formules de CLTL aux B-automates : étant donnée une formule ϕ de CLTL, on construit un B-automate $\mathcal{A}_\phi$ tel que $[\![\phi]\!] \approx [\![\mathcal{A}_\phi]\!]$. On va en fait faire mieux et obtenir $[\![\phi]\!] = [\![\mathcal{A}_\phi]\!]$, sous réserve d'autoriser des B-actions non-atomiques sur les transitions, comme par exemple `icricic`. On peut contracter ces actions en actions atomiques (ici `r`), ce qui résulte en l'affaiblissement de $[\![\phi]\!] = [\![\mathcal{A}_\phi]\!]$ en $[\![\phi]\!] \approx [\![\mathcal{A}_\phi]\!]$.

Cette construction est une généralisation de la traduction classique de LTL vers les automates de Büchi, décrite dans [DG10]. Il s'agit de garder l'intuition de cette transformation, tout en prenant en compte les opérateurs quantitatifs $\mathbf{U}^{\leq N}$ et leur signification. On travaille ici sur mots finis, donc la condition de Büchi est remplacé par celle d'état final, mais ceci n'a pas beaucoup de conséquences sur la construction.

Soit ϕ une formule de CLTL. On définit $\mathrm{sub}(\phi)$ comme l'ensemble des sous-formules de ϕ. Soit $Q = 2^{\mathrm{sub}(\phi)}$ l'ensemble des parties de $\mathrm{sub}(\phi)$.

On va définir le B-automate $\mathcal{A}_\phi = \langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta \rangle$.

L'idée directrice est qu'un état représente l'ensemble des sous-formules que l'on cherche à satisfaire avant la fin du mot. Ceci justifie le choix des états initiaux $In = \{\{\phi\}\}$ et des états finaux $Fin = \{\emptyset, \{\Omega\}\}$.

On choisit comme ensemble de compteurs $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$, où k est le nombre d'opérateurs $\mathbf{U}^{\leq N}$, étiquetés $\mathbf{U}_1^{\leq N}, \mathbf{U}_2^{\leq N}, \ldots, \mathbf{U}_k^{\leq N}$.

Les définitions suivantes suivent le cas classique (avec l'exception de l'opérateur Ω), on peut se reporter à [DG10] pour plus de détails :

Définition 4.2.1

- *Une formule atomique est $a \in \mathbb{A}$ ou Ω*
- *Un ensemble Z de formules est* consistant *s'il contient au plus une formule atomique (deux formules atomiques distinctes sont toujours contradictoires).*
- *Une formule est* réduite *si c'est une formule atomique ou si elle est de la forme $X\varphi$.*
- *Un ensemble Z de formules est réduit si tous ses éléments sont réduits.*
- *Si Z est consistant et réduit, on définit* $\mathrm{next}(Z) = \{\varphi / X\varphi \in Z\}$.
- *Si Z est un ensemble de formules, on notera $\bigwedge Z$ la conjonction de toutes les formules de Z.*

Lemme 4.2.2 *Si Z est consistant et réduit, alors pour tout $u \in \mathbb{A}^*, a \in \mathbb{A}$ et $n \in \mathbb{N}$,*

$$(au, n) \models \bigwedge Z \text{ si et seulement si } (u, n) \models \bigwedge \mathrm{next}(Z) \text{ et } Z \cup \{a\} \text{ consistant}$$

Démonstration Si $(au, n) \models \bigwedge Z$, alors $Z \cup \{a\}$ est consistant, puisque soit Z contient a, soit Z ne contient pas de formule atomique. De plus, en dehors de

a, Z ne contient que des formules de la forme $\mathbf{X}\varphi$ telles que $(au, n) \models \mathbf{X}\varphi$, et donc $(u, n) \models \varphi$. On a donc bien $(u, n) \models \bigwedge \text{next}(Z)$ et $Z \cup \{a\}$ consistant.

Réciproquement, si $(u, n) \models \bigwedge \text{next}(Z)$ et $Z \cup \{a\}$ consistant, alors Z contient des formules $\mathbf{X}\varphi$ avec $(u, n) \models \varphi$, et éventuellement la formule a. On a donc bien $(au, n) \models \bigwedge Z$. □

D'après ce lemme, si Z consistant et réduit est la liste de contraintes à satisfaire à la position i, et que ces contraintes ne sont pas incompatibles avec la lettre courante a, alors next(Z) est la liste de contraintes à satisfaire à la position $i + 1$.

On veut donc définir les transitions de $\mathcal{A}_\phi$ comme étant de la forme $Z \longrightarrow \text{next}(Z)$.

Or, en général, next(Z) n'est pas consistant et réduit, et on ne peut donc pas repartir de cet état pour avancer. Si next(Z) est inconsistant, on peut le retirer de l'automate. S'il est consistant, on va lui appliquer certaines règles de réduction pour le transformer en un état réduit. Formellement, on passera donc par des états intermédiaires au moyen d'ε-transitions. Ces états intermédiaires ne seront pas des états réels de l'automate, on les appellera « pseudo-états », et ils disparaitront de la construction à la fin, lorsque l'on contractera les ε-transitions pour obtenir uniquement des transitions étiquetées par des lettres.

Soit Y un pseudo-état (ou un état réel), et ψ une formule non réduite de taille maximale dans Y. On ajoute les transitions suivantes :

- Si $\psi = \varphi_1 \wedge \varphi_2$: $Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, \varphi_2\}$
- Si $\psi = \varphi_1 \vee \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$
- Si $\psi = \varphi_1 \mathbf{U} \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, \mathbf{X}\psi\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$
- Si $\psi = \varphi_1 \mathbf{U}_j^{\leq N} \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, \mathbf{X}\psi\} \\ Y \xrightarrow{\varepsilon:\mathtt{ic}_j} Y \setminus \{\psi\} \cup \{\mathbf{X}\psi\} \text{ (on compte une erreur)} \\ Y \xrightarrow{\varepsilon:\mathtt{r}_j} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$

  où l'action $\mathtt{r}_j$ (resp. $\mathtt{ic}_j$) effectue l'action $\mathtt{r}$ (resp. $\mathtt{ic}$) sur le compteur γ_j et ε sur les autres compteurs.

A priori, les pseudo-états atteints par ces ε-transitions n'appartiennent pas à $Q = 2^{\text{sub}(\phi)}$, car les nouvelles formules de la forme $\mathbf{X}\psi$ pour $\psi \in \text{sub}(\phi)$ peuvent ne pas être des sous-formules de ϕ. En revanche, si Z est un pseudo-état réduit, alors next(Z) sera dans Q, puisque l'on a retiré les opérateurs $\mathbf{X}$ que l'on avait ajoutés. De plus, une telle séquence d'ε-transitions est toujours finie, car si T est la taille maximale d'une formule de Y, la quantité $(T, \text{card}\{\varphi : |\varphi| = T\})$ décroît strictement pour $\leq_{lex}$ à chaque ε-transition.

Les transitions de l'automate $\mathcal{A}_\phi$ sont donc définies comme suit :

$$\Delta = \left\{ Y \xrightarrow{a:\sigma} \text{next}(Z) \mid Y \in Q, Z \cup \{a\} \text{ consistant et réduit}, Y \xrightarrow{\varepsilon:\sigma}_* Z \right\}$$

où « $Y \xrightarrow{\varepsilon:\sigma}_* Z$ » signifie qu'il existe une séquence d'ε-transitions de Y vers Z avec σ comme action combinée sur les compteurs. Pour l'instant on veut préserver

la sémantique exacte, donc on ne contracte pas σ en action atomique. σ est donc un mot appartenant à $\{\mathtt{ic}, \varepsilon, \mathtt{r}\}^*$. Le lemme suivant garantit la cohérence des pseudo-états en tant qu'états intermédiaires d'une exécution :

Lemme 4.2.3 *Soit $u = a_1 \dots a_m$ un mot de $\mathbb{A}$ et $Y_0 \stackrel{a_1:\sigma_1}{\to} Y_1 \stackrel{a_2:\sigma_2}{\to} \dots \stackrel{a_m:\sigma_m}{\to} Y_m$ une exécution acceptante de $\mathcal{A}_\phi$ sur u.*

Alors pour tout $\psi \in \mathrm{sub}(\phi)$, pour tout $n \in \{0, \dots, m\}$, pour tout $Y_n \stackrel{\varepsilon:\sigma}{\to}_* Y \stackrel{\varepsilon:\sigma'}{\to}_* Z$ avec $Z \cup \{a_{n+1}\}$ consistant et réduit, et $Y_{n+1} = \mathrm{next}(Z)$

$$\psi \in Y \implies (a_{n+1}a_{n+2}\dots a_m, K) \models \psi$$

où $K = val_B(\sigma'\sigma_{n+1}\dots\sigma_m)$.

Démonstration On effectue une récurrence inversée sur n (i.e. une récurrence sur $m - n$). Si $n = m$, Y_n est final et donc $Y_n = \emptyset$ ou $Y_n = \{\Omega\}$. Si $Y_n \stackrel{\varepsilon:\sigma}{\to}_* Y$, alors $Y = Y_n$ (il n'y a pas d'ε-transitions sortant de $\emptyset$ ou $\{\Omega\}$). Donc si $\psi \in Y$, la seule possibilité est $\psi = \Omega$, mais $a_{n+1}\dots a_m = \varepsilon$, et $(\varepsilon, 0) \models \Omega$, donc le résultat est vrai pour $n = m$.

Soit $n < m$, on suppose le résultat vrai pour $n+1$, et on reprend les notations de l'énoncé du lemme, avec $\psi \in Y$. Par définition de Δ, il existe une transition $Y_n \stackrel{a_{n+1}:\sigma\sigma'}{\longrightarrow}_* \mathrm{next}(Z) = Y_{n+1}$ dans l'automate $\mathcal{A}_\phi$.

On procède maintenant par récurrence sur la longueur k du chemin $Y \stackrel{\varepsilon:\sigma'}{\to}_* Z$.

Si $k = 0$, alors $Y = Z$ est consistant and reduit, donc ψ est soit atomique, soit de la forme $\mathbf{X}\varphi$.

Si ψ est atomique, puisque $Z \cup \{a_{n+1}\}$ est consistant, alors $\psi = a_{n+1}$. dans ce cas, $(a_{n+1}\dots a_m, K) \models \psi$.

Si $\psi = \mathbf{X}\varphi$, $\varphi \in \mathrm{next}(Z) = Y_{n+1}$, alors par hypothèse de récurrence sur n, $(a_{n+2}\dots a_m, K) \models \varphi$ (K ne change pas car σ' est vide). On obtient donc $(a_{n+1}a_{n+2}\dots a_m, K) \models X\varphi$, ce qui est le résultat cherché.

Soit $k > 0$, on suppose le résultat vrai pour des chemins de longueur $k-1$, et on le montre pour k. On a $Y \stackrel{\varepsilon:\sigma'_1}{\to} Y' \stackrel{\varepsilon:\sigma'_2}{\to}_* Z$ avec $\sigma'_1\sigma'_2 = \sigma'$, et pour tout $\psi' \in Y', (a_{n+1}a_{n+2}\dots a_m, K') \models \psi'$ avec $K' = val_B(\sigma'_2\sigma_{n+1}\dots\sigma_m)$.

On considère maintenant les différents cas possibles pour l'ε-transition $Y \stackrel{\varepsilon:\sigma'_1}{\to} Y'$. Remarquons d'abord que $K = K'$ ou $K = K' + 1$, puisque $\sigma'_1 \in \{\varepsilon, \mathtt{ic}, \mathtt{r}\}$.

Soit $u_{n+1} = a_{n+1}a_{n+2}\dots a_m$. Si $\psi \in Y'$, alors $(u_{n+1}, K') \models \psi$, mais $K \geq K'$ donc $(u_{n+1}, K) \models \psi$.

Il nous reste à examiner les cas avec $\psi \notin Y'$:

- Si $\psi = \varphi_1 \wedge \varphi_2$, $\sigma'_1 = \varepsilon$, et $Y' = Y \setminus \{\psi\} \cup \{\varphi_1, \varphi_2\}$,
  alors $(u_{n+1}, K) \models \varphi_1$ et $(u_{n+1}, K) \models \varphi_2$, d'où $(u_{n+1}, K) \models \psi$.
- Les autres cas classiques avec $\sigma'_1 = \varepsilon$ sont similaires, et sont conséquences directes de la sémantique des opérateurs LTL classiques.
- Si $\psi = \varphi_1 \mathbf{U}_j^{\leq N} \varphi_2$, $\sigma'_1 = \varepsilon$ et $Y' = Y \setminus \{\psi\} \cup \{\varphi_1, \mathbf{X}\psi\}$,
  alors $K = K'$ et $(u_{n+1}, K) \models \varphi_1$ et $(u_{n+1}, K) \models \mathbf{X}\psi$, d'où $(u_{n+1}, K) \models \psi$

– Si $\psi = \varphi_1 \mathbf{U}_j^{\leq N} \varphi_2$, $\sigma'_1 = \mathtt{ic}_j$ et $Y' = Y \setminus \{\psi\} \cup \{\mathbf{X}\psi\}$,
alors $(u_{n+1}, K') \models \mathbf{X}\psi$. Si γ_j atteint K' avant le premier reset dans $\sigma'_2\sigma_{n+1} \dots \sigma_m$, alors $K = K' + 1$, et on peut conclure $(u_{n+1}, K) \models \psi$.
Si au contraire il y a strictement moins de K' erreurs de φ_1 avant la première occurrence de φ_2, alors $K = K'$: on peut autoriser une erreur de plus tout en gardant les compteurs sous la valeur K.
– Si $\psi = \varphi_1 \mathbf{U}_j^{\leq N} \varphi_2$, $\sigma'_1 = \mathtt{r}_j$ et $Y' = Y \setminus \{\psi\} \cup \{\varphi_2\}$ alors $K = K'$, et $(u_{n+1}, K') \models \varphi_2$, d'où $(u_{n+1}, K) \models \psi$.

On peut donc conclure que peu importe la longueur du chemin k, on a $(a_{n+1}a_{n+2} \dots a_m, K) \models \psi$, ce qui conclut la preuve du lemme.

□

On utilise maintenant le Lemme 4.2.3 pour montrer que l'automate $\mathcal{A}_\phi$ a bien le comportement voulu :

Soit $Y_0 \xrightarrow{a_1:\sigma_1} Y_1 \xrightarrow{a_2:\sigma_2} \dots \xrightarrow{a_m:\sigma_m} Y_m$ une exécution valide de $\mathcal{A}_\phi$ sur u de valeur $K = [\![\mathcal{A}_\phi]\!]_B$. En appliquant le Lemme 4.2.3 avec $n = 0$ et $Y = Y_0 = \{\phi\}$, on obtient $(u, K) \models \phi$. d'où $[\![\phi]\!] \leq [\![\mathcal{A}_\phi]\!]_B$.

Réciproquement, soit $n = [\![\phi]\!](u)$, alors $(u, n) \models \phi$, donc par définition de $\mathcal{A}_\phi$, il est facile de vérifier qu'il existe une exécution acceptante de $\mathcal{A}_\phi$ sur u de valeur au plus n. Il suffit de passer à chaque fois dans l'état qui décrit les sous-formules de ϕ vraies dans la position courante. Chaque compteur γ_i comptera les erreurs de l'opérateur $\mathbf{U}_i^{\leq N}$, et ne dépassera pas n car $(u, n) \models \phi$. On a donc $[\![\mathcal{A}_\phi]\!]_B \leq [\![\phi]\!]$.

On a bien montré $[\![\mathcal{A}_\phi]\!]_B = [\![\phi]\!]$, l'automate $\mathcal{A}_\phi$ calcule en réalité la valeur exacte $[\![\phi]\!]$.

Contraction des actions

Si l'on veut obtenir un B-automate tel que défini dans la Section 2.1 avec des actions atomiques sur les compteurs, on peut procéder comme suit.

On remplace les actions $\sigma \in \{\mathtt{ic}, \varepsilon, \mathtt{r}\}^*$ par le produit de leur lettres $\pi(\sigma)$, défini dans la Figure 3.1. Par exemple `icricic` sera changé en `r`. Soit K le nombre maximal d'incréments consécutifs dans les actions σ, c'est-à-dire

$$K = \max \{ val_B(\sigma) : (p, a, \sigma, q) \in \Delta \}.$$

On appelle $\mathcal{A}'$ l'automate obtenu à partir de $\mathcal{A}_\phi$ en remplaçant chaque action σ par $\sigma' = \pi(\sigma)$. Les exécutions de $\mathcal{A}_\phi$ et $\mathcal{A}'$ sont en bijection de manière évidente : seules les actions sur les compteurs changent. Soit ρ une exécution de $\mathcal{A}_\phi$ et ρ' l'exécution correspondante.

Premièrement, ρ' fait toujours moins d'incréments que ρ (le passage de σ à σ' n'ajoute jamais d'incréments), donc $val_B(\rho') \leq val_B(\rho)$.

De plus, pour passer de σ' à σ, on peut ajouter au plus K incréments avant ou après chaque reset, donc entre deux resets (ou début/fin de mot), on a au maximum $2K$ incréments pour chaque incrément ou reset présent dans σ. Si on pose $\alpha(n) = 2Kn + 2K$, on obtient donc $val_B(\rho) \leq \alpha(val_B(\rho'))$.

Soit $u \in \mathbb{A}^*$ et ρ une exécution de $\mathcal{A}_\phi$ telle que $val_B(\rho) = [\![\mathcal{A}_\phi]\!](u)$. Alors on a vu qu'il existe une exécution ρ' de $\mathcal{A}'$ avec $val_B(\rho') = val_B(\rho)$. on obtient donc $[\![\mathcal{A}']\!] \leq [\![\mathcal{A}]\!]$.

Réciproquement, si ρ' une exécution de $\mathcal{A}'$ telle que $val_B(\rho') = [\![\mathcal{A}'i]\!](u)$. Alors on a vu qu'il existe une exécution ρ de $\mathcal{A}$ avec $val_B(\rho) \leq_\alpha val_B(\rho)$. on obtient donc $[\![\mathcal{A}']\!] \preccurlyeq_\alpha [\![\mathcal{A}]\!]$.

On obtient finalement $[\![\mathcal{A}']\!] \approx [\![\mathcal{A}_\phi]\!] = [\![\phi]\!]$, et $\mathcal{A}'$ est un B-automate avec actions atomiques.

Les résultats obtenus sont résumés dans le théorème suivant :

Théorème 4.2.4 *Si φ est une formule de CLTL, on a montré que $[\![\varphi]\!]$ est reconnue par un B-automate, et donc en particulier $[\![\varphi]\!]$ est régulière. Si de plus on autorise des actions non atomiques sur les transitions, on peut construire un B-automate qui préserve la sémantique exacte de φ.*

Puisque par [Col09], on sait décider si la fonction calculée par un B-automate est bornée, ou comparer des B-automates pour la relation $\preccurlyeq$, on a le corollaire suivant :

Corollaire 4.2.5 *Si φ et ψ sonts des formules de CLTL, on peut décider si $[\![\varphi]\!]$ est bornée, et plus généralement si $[\![\varphi]\!] \preccurlyeq [\![\psi]\!]$.*

Cependant, d'après [Col09], décider si $[\![\varphi]\!] \preccurlyeq [\![\psi]\!]$ revient à avoir $[\![\varphi]\!]$ sous la forme d'un S-automate et $[\![\psi]\!]$ sous la forme d'un B-automate. Or, décider si $[\![\varphi]\!]$ est bornée est équivalent à décider si $[\![\varphi]\!] \preccurlyeq 0$, et on veut donc obtenir φ sous la forme d'un S-automate. De plus, le passage de B à S se fait en espace exponentiel, car le semigroupe de stabilisation correspondant peut avoir un nombre exponentiel d'éléments.

Pour diminuer la complexité du problème de décision $[\![\varphi]\!] \preccurlyeq [\![\psi]\!]$, il est donc intéressant de transformer directement une formule de CLTL en S-automate.

4.3 De CLTL aux S-automates

Dans cette section, on donne une traduction d'une formule de CLTL vers le modèle des S-automates. Ceci nous permettra de montrer que le problème d'existence de borne (et même de comparaison) pour les formules de CLTL est PSPACE-complet.

4.3.1 La logique $\overline{\text{CLTL}}$

Afin de définir de manière naturelle un S-automate à partir d'une formule de CLTL, on veut commencer par renverser la sémantique de cette formule.

Soit $\overline{\text{CLTL}}$ la logique définie par la grammaire suivante :

$$\varphi := a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid \ \varphi \mathbf{R}^{>N} \varphi \mid \Omega$$

où $\mathbf{R}^{>N}$ représente la négation de $\mathbf{U}^{\leq N}$.

Ainsi, la sémantique de $\mathbf{R}^{>N}$ est définie par : $(u,n,i) \models \varphi \mathbf{R}^{>N} \psi$ si pour tout $j > i$ tel que $(u,n,j) \models \psi$, il existe au moins n positions $i \leq j' < j$ telles que $(u,n,j') \models \varphi$. Les autres opérateurs ont la même sémantique que dans CLTL.

On peut constater que si φ est une formule de CLTL, alors $\neg\varphi$ est équivalente à une formule de $\overline{\text{CLTL}}$, en poussant les négations aux feuilles comme décrit dans la section précédente.

Si φ est une formule de $\overline{\text{CLTL}}$, on définit la fonction de coût $\overline{[\![\varphi]\!]}$ définie par φ par

$$\overline{[\![\varphi]\!]}(u) = \sup \{n \in \mathbb{N} : (u,n) \models \varphi\}.$$

Lemme 4.3.1 *Si φ est une formule de CLTL, alors $\overline{[\![\neg\varphi]\!]} \approx [\![\varphi]\!]$.*

Démonstration Soit φ une formule de CLTL, et $u \in \mathbb{A}^*$. Si $[\![\varphi]\!](u)$, alors pour tout $n \in \mathbb{N}$, $(u,n) \models \neg\varphi$, et donc $\overline{[\![\neg\varphi]\!]} = \infty$. Sinon, soit $n = [\![\varphi]\!](u) \in \mathbb{N}$, alors $(u,n) \models \varphi$ et $(u,n+1) \not\models \varphi$. On a donc $(u,n) \not\models \neg\varphi$ et $(u,n+1) \models \neg\varphi$, ce qui implique $\overline{[\![\neg\varphi]\!]} = n+1$.

Ceci suffit pour affirmer $\overline{[\![\neg\varphi]\!]} \approx [\![\varphi]\!]$. □

Passer d'une formule de CLTL, à sa négation dans $\overline{\text{CLTL}}$ se fait en temps linéaire, en poussant les négations aux feuilles. On peut donc se contenter de construire un S-automate à partir d'une formule de $\overline{\text{CLTL}}$.

4.3.2 De $\overline{\text{CLTL}}$ aux S-automates

Soit ϕ une formule de $\overline{\text{CLTL}}$, possédant k opérateurs $\mathbf{R}^{>N}$, que l'on étiquette $\mathbf{R}_1^{>N}, \mathbf{R}_2^{>N}, \ldots, \mathbf{R}_k^{>N}$. On peut construire de la même manière que précédemment un S-automate $\mathcal{A}_\phi$ avec compteurs $\{\gamma_1, \ldots, \gamma_k\}$ mimant le comportement de ϕ sur son entrée u.

Les états de $\mathcal{A}_\phi$ sont de nouveau $Q = 2^{\text{sub}(\phi)}$, avec comme unique état initial $\{\phi\}$. Cependant, cette fois-ci, les états finaux sont tous les états Y tels que pour tout $\varphi \in Y$, $\varphi = \Omega$ ou φ est de la forme $\varphi_1 \mathbf{R}^{>N} \varphi_2$.

Ensuite, la principale nouveauté réside dans la manière de gérer les opérateurs $\mathbf{R}^{>N}$ dans la table d'ε-transitions entre les pseudo-états.

Soit Y un pseudo-état (ou un état réel), et ψ une formule non réduite de taille maximale dans Y. Si ψ n'est pas de la forme $\varphi_1 \mathbf{R}_j^{>N} \varphi_2$, on ajoute les même transitions que dans la preuve précédente.

Si $\psi = \varphi_1 \mathbf{R}_j^{>N} \varphi_2$, on ajoute les transitions suivantes :

$$\begin{cases} Y \xrightarrow{\varepsilon:\mathtt{i}_j} Y \setminus \{\psi\} \cup \{\varphi_1, \mathbf{X}\psi\} \text{ (on compte une occurrence de } \varphi_1) \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\mathbf{X}\psi\} \\ Y \xrightarrow{\varepsilon:\mathtt{cr}_j} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$$

On peut de la même manière que précédemment construire la table de transition de $\mathcal{A}_\phi$ en contractant les diverses ε-transitions, et vérifier que le S-automate résultant calcule bien la fonction de coût $\overline{[\![\phi]\!]}$. Pour la suite, il n'est cependant pas nécessaire d'effectuer cette contraction, on peut penser à $\mathcal{A}_phi$ comme possédant encore ces ε-transitions et ces pseudo-états.

4.3.3 Semigroupe de S-actions

On va expliciter la manière dont on contracte les S-actions, au moyen d'un semigroupe de stabilisation $\mathbf{S}$ résumant la manière dont ces actions se composent. Le produit de $\mathbf{S}$ reflète donc la concaténation de deux actions. La stabilisation $\sharp$ correspond à la répétition d'une même action un grand nombre de fois, par exemple dans un cycle.

L'élément ω représente une grande valeur, obtenue en répétant l'incrément $\mathtt{i}$ un grand nombre de fois. L'élément $\bot$ représente un échec de l'exécution, c'est-à-dire que l'on a voulu effectuer u $\mathtt{cr}$ sur une petite valeur. Les éléments de $\mathbf{S}$ sont les actions $S = \{\omega, \mathtt{i}, \varepsilon, \mathtt{r}, \mathtt{cr}\omega, \mathtt{cr}, \bot\}$, ordonnés par $\omega \leq \mathtt{i} \leq \varepsilon \leq (\mathtt{r}/\mathtt{cr}\omega) \leq \mathtt{cr} \leq \bot$. Cet ordre reflète une préférence pour le S-automate : entre deux actions $\nu \leq \nu'$, il est toujours préférable de choisir ν dans tout contexte afin d'obtenir une exécution de grande valeur. Ceci explique que $\mathtt{r}$ et $\mathtt{cr}\omega$ sont incomparables : le meilleur choix peut dépendre du contexte. En effet, dans un contexte vide, $\mathtt{r}$ est préférable, mais dans le contexte $C[x] = \omega x \mathtt{cr}$, il vaut mieux choisr $\mathtt{cr}\omega$. Le produit et la stabilisation de S sont représentés dans le tableau suivant.

$\cdot$	ω	$\mathtt{i}$	ε	$\mathtt{r}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\bot$	$\cdot^\sharp$
ω	ω	ω	ω	$\mathtt{r}$	ω	$\mathtt{r}$	$\bot$	ω
$\mathtt{i}$	ω	$\mathtt{i}$	$\mathtt{i}$	$\mathtt{r}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\bot$	ω
ε	ω	$\mathtt{i}$	ε	$\mathtt{r}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\bot$	ε
$\mathtt{r}$	ω	$\mathtt{r}$	$\mathtt{r}$	$\mathtt{r}$	$\bot$	$\bot$	$\bot$	$\mathtt{r}$
$\mathtt{cr}\omega$	$\mathtt{cr}\omega$	$\mathtt{cr}\omega$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\bot$	$\mathtt{cr}\omega$
$\mathtt{cr}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\mathtt{cr}$	$\mathtt{cr}$	$\bot$	$\bot$	$\bot$	
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Si Γ est un ensemble de compteurs, alors on note $\mathbf{S}^\Gamma$ le semigroupe produit sur l'ensemble S^γ, où l'ordre et les opérations de concaténation et de stabilisation sont effectuées composante par composante.

Si $\nu \in \mathbf{S}^\Gamma$ et $\gamma \in \Gamma$, on notera ν_γ la projection de ν sur γ. Si certaines composantes ne sont pas spécifiées, la valeur par défaut est ε. Ainsi, si $\Gamma = \{1, 2\}$, on pourra noter $\mathtt{i}_1$ pour $(\mathtt{i}, \varepsilon)$ et $\mathtt{cr}\omega_2$ pour $(\varepsilon, \mathtt{cr}\omega)$.

4.3.4 Algorithme de décision en espace polynomial

Il a été montré dans [SC85] que la satisfaisabilité d'une formule de LTL classique est un problème PSPACE-complet. Pour obtenir un algorithme en espace polynomial, un automate équivalent à la formule est généré à la volée, et un chemin acceptant dans cet automate est deviné, en ne stockant que les informations sur l'état courant et les transitions disponibles. Les étiquettes des transitions peuvent être ignorées, puisque seule l'existence d'un chemin nous intéresse.

On généralise ici cette approche : il s'agit d'explorer l'automate $\mathcal{A}_\phi$ décrit ci-dessus, en le générant à la volée et non pas d'un seul coup, afin de n'occuper qu'un espace polynomial en la taille de ϕ. Le but est maintenant de décider si

la fonction $[\![\phi]\!]$ décrite par ϕ est bornée ou non, ce qui généralise le problème de satisfaisabilité de LTL. Pour ce faire, on va chercher un témoin du fait que $[\![\mathcal{A}_\phi]\!]_S$ n'est pas bornée.

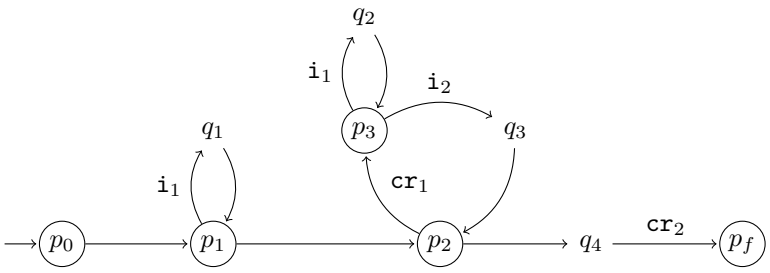
Le défi additionnel auquel on doit faire face ici est qu'il ne s'agit pas seulement de trouver une exécution acceptante de $\mathcal{A}_\phi$, mais également un témoin du fait qu'il existe des chemins acceptants de valeur arbitrairement grande. On peut donc oublier les lettres étiquetant les transitions, mais il faut prêter attention aux actions effectuées sur les compteurs.

On va devoir garder des informations sur les valeurs des compteurs tout au long du parcours. Les principes que l'algorithme doit respecter pour chaque compteur $\gamma \in \Gamma$ sont les suivants :

- Toute action $\mathtt{cr}_\gamma$ doit être précédée d'une action ω_γ, représentant un grand nombre d'incréments $\mathtt{i}_\gamma$.
- La seule manière d'obtenir ω_γ est de parcourir un cycle contenant au moins un $\mathtt{i}_\gamma$, et seulement des $\mathtt{i}$ et des ε pour γ.

Il s'agit donc de construire un algorithme non-déterministe qui devine un chemin dans l'automate, ainsi que des états que l'on visitera deux fois (de manière à créer des cycles), appelés « points de contrôle ».

On explique l'algorithme au moyen de l'exemple suivant :



Si $p_0 \in \mathit{In}$ et $p_f \in \mathit{Fin}$, la présence du chemin ci-dessus dans l'automate $\mathcal{A}_\phi$ est un exemple de témoin du fait que $[\![\mathcal{A}_\phi]\!]_S$ est non bornée. L'algorithme doit donc trouver un tel chemin, en gardant en mémoire les cycles courants et en contractant les actions rencontrées entre deux points de contrôle. Ici, les points de contrôle seront p_1, p_2 et p_3.

A tout moment, la mémoire contient une suite $m, \nu_1, p_1, \nu_2, p_2, \ldots, \nu_m, p_m$, où pour tout $i \in [1, m]$, $\nu_i \in \mathbf{S}^\Gamma$, et $p_i \in Q$. De plus, p_m représente toujours l'état courant de l'exécution que l'on veut construire, et $m - 1$ est borné par $|\Gamma|$.

Les états $(p_i)_{i<m}$ représentent des cycles ouverts mais non encore fermés. On doit les refermer en commençant par le plus récemment ouvert, de manière à obtenir des cycles imbriqués. On applique alors l'opérateur $\sharp$ à l'action du cycle, et on utilise le produit pour la concaténer avec l'action précédente.

L'algorithme commence en $0, \varepsilon, p_0$ avec $p_0 \in \mathit{In}$ (choisi de manière non-déterministe). Les transitions sont empruntées à la volée : à chaque position, on devine une transition valide (on a vu plus haut que les transitions disponibles sont données par la formule ϕ et l'état courant), et on actualise l'état mémoire. On accepte l'entrée et on renvoie « non borné » si la mémoire contient seulement

$0, \nu, p_f$, avec $p_f \in \mathit{Fin}$, et pour tout $\gamma \in \Gamma$, $\nu_\gamma \notin \{\mathtt{cr}, \mathtt{cr}\omega, \bot\}$. On peut remarquer que la condition $m - 1 \leq |\Gamma|$ limite le nombre de cycles imbriqués à $|\Gamma|$;

On reprend l'exemple d'automate donné plus haut, et on décrit dans le tableau ci-dessous l'état de la mémoire en q_1, q_2, q_3, q_4, p_f, de manière à trouver une exécution non bornée de l'automate représenté.

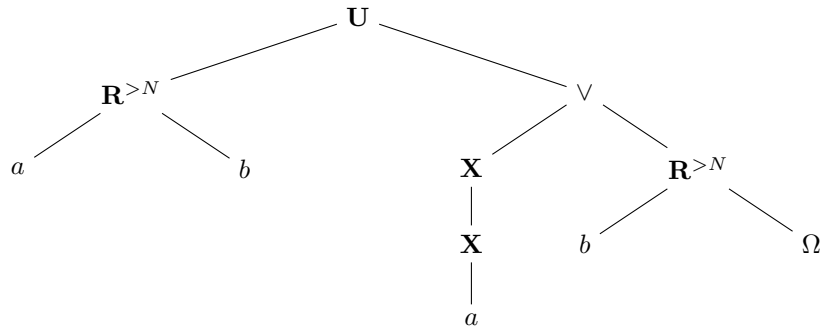
	m						
q_1	1	ε	p_1	$\mathtt{i}_1$	q_1		
q_2	2	ω_1	p_2	$\mathtt{cr}_1$	p_3	$\mathtt{i}_1$	q_2
q_3	1	ω_1	p_2	$(\mathtt{cr}\omega, \mathtt{i})$	q_3		
q_4	0	$(\mathtt{r}, \omega)$	q_4				
p_f	0	$(\mathtt{r}, \mathtt{r})$	p_f				

Après être passé en q_1, l'algorithme revient en p_1, et ferme le cycle ne contenant que l'action $\mathtt{i}_1$. L'action obtenue est donc $\mathtt{i}_1^\sharp = \omega_1$. Les points de contrôle p_2 et p_3 sont ensuite ouverts, séparés par une action $\mathtt{cr}_1$, et suivis d'une action $\mathtt{i}_1$, comme on le voit en q_2. Après l'état q_2, lorsque l'algorithme revient en p_3, l'action $\mathtt{i}_2$ est stabilisée, ce qui donne ω_2. Il faut de plus la concaténer avec le $\mathtt{cr}_1$ lu précédemment, ce qui donne l'action $(\mathtt{cr}\omega, \mathtt{i})$ présente en q_3. Ensuite, lorsque l'exécution revient sur p_2 et ferme le cycle externe, ce $(\mathtt{cr}\omega, \mathtt{i})$ est stabilisé en $(\mathtt{cr}\omega, \mathtt{i})^\sharp = (\mathtt{cr}\omega, \omega)$, et concaténé à ω_1, pour obtenir le $(\mathtt{r}, \omega)$ présent en q_4. Finalement, la concaténation avec le $\mathtt{cr}_2$ final résulte en l'action globale $(\mathtt{r}, \mathtt{r})$ présente en p_f. D'après la condition d'acceptation, on peut alors renvoyer « non borné » puisque $\mathtt{r} \notin \{\mathtt{cr}, \mathtt{cr}\omega, \bot\}$ et $p_f \in \mathit{Fin}$.

4.3.5 Complexité et correction de l'algorithme

Lemme 4.3.2 *L'algorithme décrit ci-dessus a une complexité spatiale polynomiale en $|\phi|$.*

Démonstration On commence par préciser la manière dont la formule ϕ est codée. Une telle formule peut être représentée par un arbre, dont les noeuds sont les opérateurs, et les feuilles sont les atomes. Par exemple la formule $(a\mathbf{R}^{>N}b)\mathbf{U}((\mathbf{XX}a) \vee (b\mathbf{R}^{>N}\Omega))$ peut être représentée par l'arbre suivant :



Ainsi, chaque sous-formule de ϕ correspond à un noeud de l'arbre. Un ensemble de sous-formules est donc un ensemble de noeuds, et chaque état peut être codé par un tuple $(n_1, n_2, \ldots, n_t)$, où chaque n_i code l'emplacement d'un noeud de ϕ (il est facile de voir que coder un tel emplacement est toujours polynomial en la taille totale de l'arbre). Le code d'un état occupe donc un espace polynomial en la taille de l'arbre d'entrée, notée $|\phi|$. On peut de plus remarquer que c'est toujours vrai si on ajoute aussi les pseudo-états, car ceux-ci sont des parties de $\mathrm{sub}(\phi) \cup \{\mathbf{X}\varphi : \varphi \in \mathrm{sub}(\phi)\}$. Le code d'un pseudo-état est donc au pire deux fois plus long que celui d'un état réel, et occupe toujours un espace polynomial.

Le codage d'un élément de $|\mathbf{S}|$ occupe un espace constant, donc il suffit d'un espace linéaire en $|\Gamma|$ pour coder un élément de $\mathbf{S}^\Gamma$. Or, chaque compteur de $\mathcal{A}_\phi$ est issu d'un opérateur $\mathbf{R}^{>N}$ de ϕ, donc $|\Gamma| \leq |\phi|$. En conséquence, chaque élément de $\mathbf{S}^\Gamma$ occupe un espace linéaire en $|\phi|$.

Finalement, $m \leq |\Gamma|$, donc il suffit d'un espace logarithmique en $|\phi|$ pour stocker m, et on aura toujours au plus m pseudo-états et m éléments de $\mathbf{S}^\Gamma$, chacun occupant un espace polynomial en $|\phi|$. La totalité de la suite occupe donc un espace polynomial en $|\phi|$, ce qui conclut la preuve. □

Lemme 4.3.3 *L'algorithme est correct, c'est-à-dire qu'il renvoie « non borné » si et seulement si $[\![\phi]\!]$ est non bornée.*

Démonstration Il est facile de montrer que si l'algorithme renvoie « non borné », alors $[\![\phi]\!]$ est non bornée. En effet, l'algorithme décrit un chemin (comportant des cycles) dans $\mathcal{A}_\phi$. Il est direct de montrer que si l'on emprunte chaque cycle n fois, l'exécution résultante a valeur au moins n. Le chemin décrit par l'algorithme décrit donc une famille d'exécutions, témoignant du $[\![\mathcal{A}_\phi]\!]_S$ n'est pas bornée. Puisque $[\![\mathcal{A}_\phi]\!]_S \approx [\![\phi]\!]$, on peut en conclure que $[\![\phi]\!]$ n'est pas bornée.

On montre maintenant la réciproque : on suppose que $[\![\mathcal{A}_\phi]\!]$ n'est pas bornée, et on veut montrer qu'il existe un chemin témoin qui peut être découvert par l'algorithme.

Dans ce but, on définit pour tout S-automate $\mathcal{A}$ le semigroupe de stabilisation $\mathbf{S}_\mathcal{A} = \langle S_\mathcal{A}, \cdot, \sharp, \leq \rangle$, dont les éléments représentent des ensembles exécutions partielles de $\mathcal{A}$. Cette construction reprend celle de [Col09]. On représentera une une exécution de p vers q effectuant l'action globale ν par l'élément $(p, \nu, q) \in Q \times S^\Gamma \times Q$.

Si (p, ν, q) et (p', ν', q') sont deux éléments de $Q \times S^\Gamma \times Q$, on dira que $(p, \nu, q) \leq (p', \nu', q')$ si $(p, q) = (p, q')$ et $\nu \leq \nu'$, pour l'ordre défini composante par composante sur S^Γ.

Si $E \subseteq Q \times S^\Gamma \times Q$, on notera $E\!\downarrow\! = \{e \leq e' : e' \in E\}$ la clôture vers le bas de E.

Soit $S_\mathcal{A} = 2^{Q \times S^\Gamma \times Q}\!\downarrow$ l'ensemble des éléments de $\mathbf{S}_\mathcal{A}$ clos vers le bas. Chaque élément E de $S_\mathcal{A}$ représente donc un ensemble d'exécutions possibles. L'opération de clôture vers le bas revient à considérer que l'automate a le droit d'effectuer

des actions moins bonnes que celles disponibles en réalité : cela ne change pas la sémantique globale.

Le produit et la stabilisation de $\mathbf{S}_{\mathcal{A}}$ sont définis par :

$$E \cdot F = \{(p, act_1 \cdot \nu_2, r) : (p, \nu_1, q) \in E, (q, \nu_2, r) \in F\}\downarrow$$
$$E^\sharp = \left\{(p, \nu_1 \cdot \nu_e^\sharp \cdot \nu_2, r) : (p, \nu_1, q), (q, \nu_e, q), (q, \nu_2, r) \in E\right\}\downarrow .$$

On définit également l'idéal de reconnaissance

$$I = \{E \in S_{\mathcal{A}} : \exists (p, \nu, q) \in E, p \in In, q \in Fin, \forall \gamma \in \Gamma, \nu_\gamma \notin \{\mathtt{cr}, \mathtt{cr}\omega, \bot\}\},$$

ainsi que le morphisme $h : \mathbb{A} \to \mathbf{S}_{\mathcal{A}}$ par $h(a) = \{(p, \nu, q) : (p, a, \nu, q) \in \Delta_{\mathcal{A}}\}\downarrow$.

D'après [Col09], $S_{\mathcal{A}}, h, I$ reconnaît la fonction de coût $[\![\mathcal{A}]\!]_S$. Par conséquent $[\![\mathcal{A}]\!]_S$ est non bornée si et seulement si $\langle h(\mathbb{A})\rangle^\sharp \cap I \neq \emptyset$.

On applique maintenant cette construction à l'automate $\mathcal{A}_\phi$ obtenu à partir de ϕ. Puisque l'on a supposé que $[\![\mathcal{A}_\phi]\!]$ est non bornée, il existe une $\sharp$-expression e bien formée pour $\mathbf{S}_{\mathcal{A}_\phi}$, telle que $\mathrm{eval}(e) \in I$. Il reste à montrer que e n'a pas besoin de contenir plus de $|\Gamma|$ stabilisations imbriquées.

Supposons que e contient au moins $k = |\Gamma| + 1$ stabilisations imbriquées $\sharp_1, \ldots, \sharp_k$, appliquées respectivement à des $\sharp$-expression $e_1, \ldots, e_k \in \mathbf{S}_{\mathcal{A}}$. Soit $(p_0, \nu_f, p_f) \in \mathrm{eval}(e)$, témoignant que $\mathrm{eval}(e) \in I$, c'est-à-dire avec $p_0 \in In, p_f \in Fin$, et pour tout$\gamma \in \Gamma, \nu_\gamma \notin \{\mathtt{cr}, \mathtt{cr}\omega, \bot\}$.

On dira qu'une stabilisation $\sharp_i$ est *utile* si la $\sharp$-expression e' obtenu à partir de e en retirant $\sharp_i$ ne contient pas (p_0, ν_f, p_f). On montre par récurrence sur $|\Gamma|$ que l'une des stabilisations $\sharp_1, \ldots, \sharp_k$ est inutile. Si $|\Gamma| = 0$, toute stabilisation est inutile. On suppose maintenant $|\Gamma| \geq 1$.

Soit $\sharp_k$ la stabilisation la plus extérieure dans e. Ainsi $e = x \cdot e_k^{\sharp_k} \cdot y$, où x et$y$ sont des $\sharp$-expression. On pose $E = \mathrm{eval}(e) = \mathrm{eval}(x)\mathrm{eval}(e_k^\sharp)\mathrm{eval}(y) = XE_k^s harpY$. On suppose que $\sharp_k$ est utile (sinon on a le résultat voulu). Par définition du produit, seulement l'un des éléments de $E_k^\sharp$ est utilisé pour obtenir $(p_0, \nu_f, p_f) \in E$. Par définition de $\sharp$, cet élément est de la forme (p, ν, r), avec $(p, \nu_1, q), (q, \nu_e, q), (q, \nu_2, r) \in E$ et $\nu \leq \nu_1 \cdot \nu_e^\sharp \cdot \nu_2$. Puisque $\sharp_k$ est utile, on doit avoir $\nu_e^\sharp \neq \nu_e$, et donc il existe $\gamma \in \Gamma$ tel que $(\nu_e)_\gamma = \mathtt{i}$. De plus, d'après les opérations sur S, e_k ne peut contenir aucune stabilisation utile du compteur γ. On est donc ramené aux $k-1$ stabilisations présentes dans e_k, et $|\Gamma| - 1$ compteurs disponibles, puisque γ n'est plus concerné par les stabilisations. Ceci conclut la preuve par récurrence.

On peut en conclure que e est équivalente à une $\sharp$-expression e' contenant au plus $|\Gamma|$ stabilisations imbriquées. Le fait que $\mathrm{eval}(e') \in I$ nous garantit l'existence d'un chemin dans $\mathcal{A}_\phi$ qui peut être trouvé par l'algorithme, puisqu'il possède au plus $|\Gamma|$ cycles imbriqués. Ceci conclut la preuve du Lemme. □

Théorème 4.3.4 *Etant donné une formule de CLTL ϕ, le problème de savoir si $[\![\phi]\!]$ est bornée est PSPACE-complet.*

Démonstration On a vu qu'on peut obtenir un algorithme PSPACE pour répondre à cette question. On commence par nier ϕ pour obtenir une formule ϕ'

de $\overline{\text{CLTL}}$, ceci se fait en temps linéaire. On dispose ensuite d'une description de $\mathcal{A}_{\phi'}$ qui nous permet de le parcourir à la volée en espace polynomial, de manière à trouver un chemin témoignant de $[\![\phi]\!]$ non borné, si un tel chemin existe.

Pour montrer que le problème est PSPACE-difficile, il suffit de remarquer que le problème de satisfaisabilité de LTL est un cas particulier de celui-ci, et est lui-même PSPACE-difficile d'après [SC85]. En effet, si ϕ est une formule de LTL classique, on peut voir $\neg\phi$ comme une formule de CLTL, et dans ce cas, « $[\![\neg\phi]\!]$ bornée » est équivalente à $L(\phi) = \emptyset$. □

On a montré que la généralisation de LTL en CLTL n'augmente pas la complexité de la vérification des formules. Ce résultat est encourageant, puisque l'on parvient à traiter un cas plus général, sans contrepartie en termes de ressources algorithmiques.

4.4 Caractérisation algébrique

On s'attache maintenant à étudier le fragment défini par CLTL. On cherche en particulier à obtenir la décidabilité de l'appartenance à ce fragment : étant donné une fonction de coût régulière f, peut-on décider s'il existe une fomule φ de CLTL reconnaissant f ? Dans le cas des langages réguliers, ce problème peut être décidé grâce à l'équivalence entre LTL, les expressions régulières sans étoiles, et les semigroupes apériodiques [DG10, Sch65]. Malgré le fait qu'on ne dispose pas ici des expressions régulières, on va généraliser ce résultat aux fonctions de coût, en montrant directement l'équivalence entre CLTL et les semigroupes de stabilisation apériodiques.

Si f est une fonction de coût régulière, on notera $\mathbf{S}_f$ son semigroupe de stabilisation minimal.

La définition suivante généralise aux semigroupes de stabilisation la notion classique d'apériodicité.

Définition 4.4.1 *Un semigroupe de stabilisation fini* $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ *est dit* apériodiqe *si*

$$\exists k \in \mathbb{N}, \forall s \in \mathbf{S}, s^{k+1} = s^k.$$

On peut remarquer qu'un semigroupe de stabilisation est apériodique s'il ne contient aucun groupe non trivial. Intuitivement, l'apériodicité est une propriété structurelle qui empêche par exemple de compter modulo k (pour tout entier k). Par exemple il est facile de montrer que la fonction de l'Exemple 2.3.14 ne peut pas être reconnue par un semigroupe apériodique.

Proposition 4.4.2 *L'apériodicité est préservée par quotient de semigroupes, et donc en particulier de semigroupes de stabilisation.*

Pour une fonction de coût régulière f, les assertions suivantes sont donc équivalentes :

- *Il existe un semigroupe de stabilisation apériodique qui reconnaît f,*
- *$\mathbf{S}_f$ est apériodique.*

On va maintenant montrer le théorème suivant, qui caractérise algébriquement, au moyen des semigroupes apériodiques, les fonctions de coût reconnues par une formule de CLTL.

Théorème 4.4.3 *Soit f une fonction de coût régulière, les assertions suivantes sont équivalentes :*
- *il existe une formule φ de CLTL telle que $f \approx [\![\varphi]\!]$,*
- *$\mathbf{S}_f$ est apériodique.*

4.4.1 De CLTL à l'apériodicité

On montre ici la première implication du Théorème 4.4.3, c'est-à-dire le théorème suivant :

Théorème 4.4.4 *Soit φ une formule de CLTL, alors $[\![\varphi]\!]$ est régulière d'après le Théorème 4.2.4, et $\mathbf{S}_f$ est apériodique.*

Démonstration

La preuve de ce théorème va montrer l'utilité de la congruence syntactique définie dans la Section 2.3.5.

L'idée générale est d'utiliser les $\omega\sharp$-expressions, et la relation $\equiv_f$, pour décrire directement le semigroupe minimal. Plus précisément, d'après le Théorème 2.3.10 : $\mathbf{S}_f$ est apériodique s'il existe $k \in \mathbb{N}$ tel que pour toute $\omega\sharp$-expression E, $E^k \equiv_f E^{k+1}$.

La preuve procède par induction sur ϕ.

Cas $\phi = a$

On a $S_{[\![\phi]\!]} = \{a, b\}$ avec $a \cdot b = a \cdot a = a$, et $b \cdot a = b \cdot b = b$, donc $\mathbf{S}_{[\![\phi]\!]}$ est apériodique (c'est aussi vrai pour $\phi = \neg a$).

Cas $\phi = \Omega$

Alors $S_{[\![\phi]\!]} = \{1, a\}$ avec 1 élément neutre, et $a \cdot a = a$, $\mathbf{S}_{[\![\phi]\!]}$ est donc apériodique.

Cas $\phi = \varphi_1 \wedge \varphi_2$ ou $\phi = \varphi_1 \vee \varphi_2$

Il est facile de vérifier que ϕ est reconnue par le semigroupe produit $\mathbf{S}_{[\![\varphi_1]\!]} \times \mathbf{S}_{[\![\varphi_2]\!]}$, en prenant pour I l'union ou l'intersection des I_1 et I_2 correspondants. $\mathbf{S}_{[\![\varphi_1]\!]}$ et $\mathbf{S}_{[\![\varphi_2]\!]}$ sont apériodiques par hypothèse d'induction, et l'apériodicité est préservée par produit, donc $\mathbf{S}_{[\![\phi]\!]}$ est apériodique.

Case $\phi = \mathbf{X}\psi$

Par hypothèse d'induction, $\mathbf{S}_{[\![\psi]\!]}$ est apériodique, donc il existe $k \in \mathbb{N}$ tel que pour toute $\omega\sharp$-expression E, $E^k \equiv_{[\![\psi]\!]} E^{k+1}$. On veut montrer que c'est vrai aussi pour $[\![\phi]\!]$.

On reprend la notation $\underset{n\to\infty}{\bowtie}$ définie dans la preuve du Lemme 2.3.18 (Lemme de Désynchronisation) : Si a_n et b_n sont deux expressions qui contiennent la variable libre n, et à valeur dans $\mathbb{N}$, on dira que $a_n \underset{n\to\infty}{\bowtie} b_n$ si $\limsup a_n = \infty \Leftrightarrow \limsup b_n = \infty$.

Soit E une $\omega\sharp$-expression, et $e = E[\omega \leftarrow \max(K_{[\![\phi]\!]}!, K_{[\![\psi]\!]}!)]$, où les constantes K sont donnés par le Lemme 2.3.3. e est donc une $\sharp$-expression bien formée dans $\mathbf{S}_{[\![\psi]\!]}$ et dans $\mathbf{S}_{[\![\phi]\!]}$.

On veut montrer que $E^{k+2} \equiv_{[\![\phi]\!]} E^{k+1}$, c'est-à-dire pour tout contexte $C[x]$,

$$[\![\phi]\!](C[e^{k+2}](n)) \underset{n\to\infty}{\bowtie} [\![\phi]\!](C[e^{k+1}](n)).$$

Soit $C[x]$ un contexte.

– Si $C[x] = aC'[x]$, alors par la proposition 2.3.16 avec contexte xe :

$$\begin{aligned}[\![\phi]\!](C[e^{k+2}](n)) &= [\![\psi]\!](C'[e^{k+2}](n)) \\ &\underset{n\to\infty}{\bowtie} [\![\psi]\!](C'[e^{k+1}](n)) \\ &= [\![\phi]\!](C[e^{k+1}](n)).\end{aligned}$$

– Si le début de $C[x]$ est une lettre a sous au moins un $\sharp$, alors il existe un contexte $C'[x]$ tel que pour toute $\sharp$-expression e' et tout $n \in \mathbb{N}$, $C[e'](n+1) = aC'[e'](n)$. Par exemple si $C[x] = ((ax)^\sharp b)^\sharp$ alors $C'[x] = x(ax)^\sharp b((ax)^\sharp b)^\sharp$. On peut donc écrire

$$\begin{aligned}[\![\phi]\!](C[e^{k+2}](n+1)) &= [\![\psi]\!](C'[e^{k+2}](n)) \\ &\underset{n\to\infty}{\bowtie} [\![\psi]\!](C'[e^{k+1}](n)) \\ &= [\![\phi]\!](C[e^{k+1}](n+1)).\end{aligned}$$

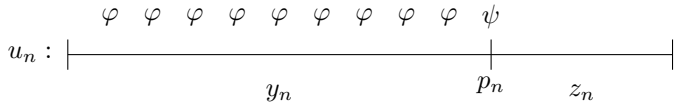
– Finalement, si $C[x]$ commence par x (possiblement sous un ou plusieurs $\sharp$), on remplace x par ex dans $C[x]$, pour se ramener au cas précédent. On obtient ainsi un autre contexte $C'[x]$ tel que $C[e^{k+1}](n+1) = aC'[e^k](n)$ et $C[e^{k+2}](n+1) = aC'[e^{k+1}](n)$ pour tout n, d'où

$$\begin{aligned}[\![\phi]\!](C[e^{k+2}](n+1)) &= [\![\phi]\!](aC'[e^{k+1}](n)) \\ &= [\![\psi]\!](C'[e^{k+1}](n)) \\ &\underset{n\to\infty}{\bowtie} [\![\psi]\!](C'[e^k](n)) \\ &= [\![\phi]\!](aC'[e^k](n)) \\ &= [\![\phi]\!](C[e^{k+1}](n+1))\end{aligned}$$

Cas $\phi = \varphi\mathbf{U}\psi$

On sait par hypothèse d'induction que $\mathbf{S}_{[\![\varphi]\!]}$ et $\mathbf{S}_{[\![\psi]\!]}$ sont apériodiques, donc il existe $k \in \mathbb{N}$ tel que pour tout $\omega\sharp$-expression E, $E^k \equiv_{[\![\varphi]\!]} E^{k+1}$ et $E^k \equiv_{[\![\psi]\!]} E^{k+1}$. Soit E une $\omega\sharp$-expression. on veut montrer que $E^{k+1} \equiv_{[\![\phi]\!]} E^{k+2}$.

Soit $C[x]$ un contexte, $K = \max(K_{[\![\varphi]\!]}, K_{[\![\psi]\!]})$ (ces constantes étant données par le Lemme 2.3.3), $u_n = C[E^{k+1}](K!, n)$ et $v_n = C[E^{k+2}](K!, n)$. On veut montrer que $C[E^{k+1}] \rightleftharpoons_{[\![\phi]\!]} C[E^{k+2}]$, c'est-à-dire $[\![\phi]\!](u_n) \underset{n\to\infty}{\bowtie} [\![\phi]\!](v_n)$. Supposons par exemple que $[\![\phi]\!](u_n)$ est bornée par m On a alors $(u_n, m) \models \phi$ pour tout n. D'après la sémantique de l'opérateur $\mathbf{U}$, pour tout n il existe p_n tel que $(u_n, m, p_n) \models \psi$ et pour tout $i \in [0, p_n - 1]$, $(u_n, m, i) \models \varphi$. Pour tout n, on définit z_n le suffixe de u_n commençant en position p, et y_n^i le facteur de u_n contenant les positions $[i, p_n - 1]$. On pose finalement $y_n = y_n^0$, ainsi $u_n = y_n z_n$.



On examine maintenant la position p_n pour chaque n. Cette position correspond toujours à une position q_n dans la $\sharp$-expression $e = C[E^{k+1}](K!)$. Cependant e est finie, donc on peut extraire une suite constante $q_{\delta(n)}$ de q_n. Soit p la valeur de cette suite constante, p est une position dans e. Soit $\{e_j, j \in J\}$ l'ensemble fini de sous-$\sharp$-expression de e telles que $e_j^\sharp$ contiennent la position p (la relation de sous-expression les ordonne totalement). On peut choisir $J = [1, r]$ tel que $1 \leq j < j' \leq r$ implique $e_j^\sharp$ est une sous-expression de $e_{j'}$.

On étiquette également chaque opérateur $\sharp$ de $e_j^\sharp$ par j. Remarquons que J peut être vide, si p ne se trouve pas sous un $\sharp$ dans e.

On note $\overleftarrow{f_j}(\delta(n))$ le nombre d'occurrences de $e_j(\delta(n))$ (venant de $e_j^\sharp$ correspondant) dans $y_{\delta(n)}$. On définit de la même manière $\overrightarrow{f_j}(\delta(n))$ par rapport à $z_{\delta(n)}$.

On obtient pour tout $n \in \mathbb{N}$, $\delta(n) - 1 \leq \overleftarrow{f_j}(\delta(n)) + \overrightarrow{f_j}(\delta(n)) \leq \delta(n)$. La borne inférieure $\delta(n) - 1$ est due au fait que p peut se trouver au milieu d'une occurrence de e_j, qui ne sera comptée ni dans $y_{\delta(n)}$ ni dans $z_{\delta(n)}$.

Ceci implique que pour tout $j \in J$, on se trouve dans l'un de ces 3 cas :

- $j \in J_1$: $\overleftarrow{f_j}(\delta(n))$ n'est pas borné et $\overrightarrow{f_j}(\delta(n))$ est borné.
- $j \in J_2$: $\overleftarrow{f_j}(\delta(n))$ est borné $\overrightarrow{f_j}(\delta(n))$ n'est pas borné.
- $j \in J_3$: $\overleftarrow{f_j}(\delta(n))$ et $\overrightarrow{f_j}(\delta(n))$ sont tous les deux non bornés.

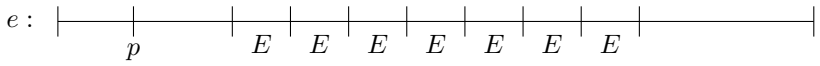
Or, J est fini, donc on peut extraire $\sigma(n)$ de $\delta(n)$, et partitionner J en $J_1 \uplus J_2 \uplus J_3$ tel que pour tout $j \in J$:

- Si $j \in J_1$, $\overleftarrow{f_j}(\sigma(n)) \to \infty$ et $\overrightarrow{f_j}(\sigma(n))$ est constant.
- Si $j \in J_2$, $\overleftarrow{f_j}(\sigma(n))$ est constant $\overrightarrow{f_j}(\sigma(n)) \to \infty$.
- Si $j \in J_3$, $\overleftarrow{f_j}(\sigma(n)) \to \infty$ et $\overrightarrow{f_j}(\sigma(n)) \to \infty$.

Remarquons que si $j < j'$ et $\overrightarrow{f_j} \circ \sigma \neq 0$, alors $j \notin J_1$. De manière symétrique, si $j < j'$ et $\overleftarrow{f_j} \circ \sigma \neq 0$, alors $j \notin J_2$.

On peut maintenant distinguer 3 cas pour la position p dans $e = C[E^{k+1}](K!)$:

Premier cas : p est avant la première occurrence de E dans e.



On considère alors le contexte $C'[x]$ obtenu en remplaçant dans $C[x]$ chaque opérateur $\sharp_j$ par la valeur constante de $\overrightarrow{f_j}(\sigma(n))$ pour tout $j \in J_1$. Ce passage de C à C' permet d'évaluer le mot en partant de la position p, et non plus du début. On a $[\![\psi]\!](z_{\sigma(n)}) \leq m$ pour tout n, mais par le Lemme de Désynchronisation, $[\![\psi]\!](z_n)$ est borné si et seulement si $C'[E^{k+1}] \in [\![\psi]\!]^B$. Par hypothèse d'induction, $C'[E^{k+1}] \in [\![\psi]\!]^B \Leftrightarrow C'[E^{k+2}] \in [\![\psi]\!]^B$. Soit z'_n le suffixe de $C[E^{k+2}](K!, n)$ commençant en position p_n. En réutilisant le Lemme de Désynchronisation, on obtient $[\![\psi]\!](z'_{\sigma(n)}) \leq m'$ pour un certain m' (R).

Il reste à montrer qu'il existe une constante M telle que $[\![\varphi]\!](y^i_{\sigma(n)} z'_{\sigma(n)}) \leq M$ pour tout $n \in \mathbb{N}$ et $i \in [0, p_{\sigma(n)}]$ (les $y^i_{\sigma(n)}$ ne sont pas affectés par le changement de E^{k+1} en E^{k+2}).

On notera $g^i_{\sigma(n)} = [\![\varphi]\!](y^i_{\sigma(n)} z'_{\sigma(n)})$ pour la lisibilité. Supposons qu'un tel M n'existe pas, alors $\left\{ g^i_{\sigma(n)} : n \in \mathbb{N}, 1 \leq i \leq p_{\sigma(n)} \right\}$ n'est pas borné. On va utiliser un argument diagonal pour arriver à une contradiction.

Pour tout n, soit $i_{\sigma(n)}$ tel que $g^{i_{\sigma(n)}}_{\sigma(n)} = \max \left\{ g^i_{\sigma(n)} : 1 \leq i \leq p_{\sigma(n)} \right\}$. Par construction, la suite $g^{i_{\sigma(n)}}_{\sigma(n)} = [\![\varphi]\!](y^{i_{\sigma(n)}}_{\sigma(n)} z'_{\sigma(n)})$ n'est pas bornée.On peut donc extraire $\sigma'(n)$ de $\sigma(n)$ tel que $g^{i_{\sigma'(n)}}_{\sigma'(n)} \to \infty$.

On peut maintenant répter le processus utilisé précédemment pour extraire $\gamma(n)$ de $\sigma'(n)$, tel que pour tout n, les positions de départ des $y^{i_{\gamma(n)}}_{\gamma(n)}$ correspondent toutes à la même position dans e, et tel qu'il existe un contexte $C''[x]$ avec $[\![\varphi]\!](y^{i_{\gamma(n)}}_{\gamma(n)} z_{\gamma(n)}) \underset{n\to\infty}{\bowtie} [\![\varphi]\!](C''[E^{k+1}](K!, \gamma(n)))$ (de nouveau par le Lemme de Désynchronisation).

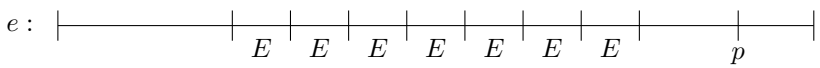
On peut maintenant ajouter une occurrence de E (passant ainsi de $k+1$ à $k+2$ E consécutifs) et changer ainsi z en z' (les facteurs y ne sont pas concernés par les occurrences de E). On obtient $g^{i_{\gamma(n)}}_{\gamma(n)} \underset{n\to\infty}{\bowtie} [\![\varphi]\!](C''[E^{k+2}](K!, \gamma(n)))$.

Par hypothèse, $[\![\varphi]\!](y^{i_{\gamma(n)}}_{\gamma(n)} z_{\gamma(n)})$ est borné par m, et $C''[E^{k+1}] \rightleftharpoons_{[\![\varphi]\!]} C''[E^{k+2}]$, donc $g^{i_{\gamma(n)}}_{\gamma(n)}$ est borné. Or, on sait que $g^{i_{\gamma(n)}}_{\gamma(n)} \to \infty$. On aboutit à une contradiction, ce qui prouve l'existence d'un M.

On a montré qu'il existe un entier M tel que pour tout $n \in \mathbb{N}$ et $i \in [0, p_n]$, $[\![\varphi]\!](y^i_{\sigma(n)} z'_{\sigma(n)}) \leq M$. En combinant ceci avec le résultat (R) obtenu sur ψ, on obtient $[\![\varphi U \psi]\!](C[E^{k+2}](K!, n)) \leq \max(m', M)$. On a donc $C[E^{k+1}] \in [\![\phi]\!]^B \implies C[E^{k+2}] \in [\![\phi]\!]^B$.

La direction inverse est similaire, il suffit de supprimer une occurrence de E au lieu d'en ajouter une. On obtient donc $C[E^{k+1}] \rightleftharpoons_{[\![\phi]\!]} C[E^{k+2}]$.

Second cas : p est après la dernière occurrence de E dans e.



Cette fois z_n ne sera pas affecté par le passage de E^{k+1} à E^{k+2}, mais les y_n^i le seront. Soit $y_n'^i z_n$ les suffixes de $v_n = C[E^{k+2}](K!, n)$, et p'_n la position de départ de z_n dans v_n (on a donc $i \in [0, p_n - 1]$). Comme précédemment, on suppose que $\left\{ [\![\varphi]\!](y'^i_{\sigma(n)} z_{\sigma(n)}) : n \in \mathbb{N}, 1 \leq i \leq p_{\sigma(n)} \right\}$ n'est pas borné, et on construit une suite $y'^{i_{\gamma(n)}}_{\gamma(n)}$ avec une position de départ constante dans e, et telle que $[\![\varphi]\!](y'^i_{\gamma(n)} z_{\gamma(n)}) \to \infty$. On pose pour tout $n \in \mathbb{N}$, $w_n = y'^i_{\gamma(n)} z_{\gamma(n)}$.

De la même manière, on extrait un contexte $C''[x]$, mais cette fois-ci il est nécessaire d'utiliser une fois de plus le Lemme de Désynchronisation 2.3.18, pour associer à chaque opérateur $\sharp_j$ de $C''[x]$ la valeur $\overrightarrow{f_j}(\sigma(n))$ au lieu de $\sigma(n)$.

L'idée est d'associer des positions de $v_{\gamma(n)}$ à des positions de $u_{\gamma(n)}$ de manière à utiliser ce que l'on sait du comportement de $u_{\gamma(n)}$ pour borner les valeurs $[\![\varphi]\!](w_n)$, et ainsi obtenir une contradiction. Trois cas sont à distinguer :

- Si un facteur correspondant à E^{k+2} apparaît dans w_n, alors la preuve précédente est toujours valide, et on peut associer w_n à un certain $y^{j_{\gamma(n)}}_{\gamma(n)} z_{\gamma(n)}$ ($j_{\gamma(n)}$ peut différer de $i_{\gamma(n)}$) pour obtenir la contradiction. Cette association doit juste prendre en compte le décalage dû aux nouvelles occurrences de E, mais les positions dans les mots sont essentiellement les mêmes.
- Si E apparaît au plus k fois dans w_n, alors on peut associer aux positions de v_n celles de u_n en prenant en compte le décalage du aux occurrences de E en début de mot, et obtenir la contradiction sans avoir besoin d'utiliser l'hypothèse de récurrence.
- Sinon, E apparaît $k+1$ fois dans w_n, et on peut utiliser l'équivalence $E^{k+1} \equiv_{[\![\varphi]\!]} E^k$ pour associer les positions de v_n avec des positions de u_n et obtenir la contradiction désirée. Cette fois les positions du premier E de chaque séquence E^k dans v_n sera associé au second dans u_n. Informellement, on « duplique » le premier E de chaque séquence.

En résumé, la figure suivante montre la manière dont on associe les positions de v_n et celles de u_n, avec $k = 2$. Cette figure est juste un exemple, elle est plus simple que le cas général, car il n'y a ici qu'une seule séquence de E. Si une autre séquence est présente avant, elle décale tout ce schéma, mais le principe reste le même.

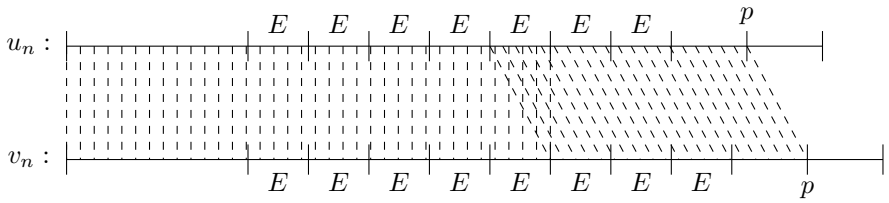
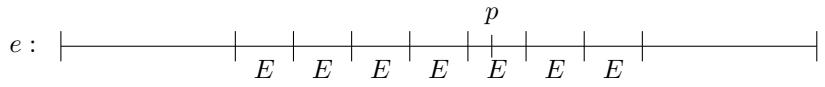


FIGURE 4.1 – Association de positions

Ce choix d'association est arbitraire : on peut en fait choisir comme on veut le E à dupliquer, on pourra dans tous les cas utiliser l'hypothèse d'induction sur u_n, ou juste l'hypothèse selon laquelle φ est vraie sur les suffixes de u_n avant p, pour conclure.

Troisième cas : p peut se trouver dans une séquence de E.



On a en fait déjà décrit toutes les techniques permettant de résoudre ce cas. La seule nouveauté est le fait que ψ est maintenant également concerné par le changement du nombre d'occurrence de E, et on aura donc besoin de l'hypothèse $E^{k+1} \equiv_{[\![\psi]\!]} E^k$, si $z'_{\sigma(n)}$ contient $k+1$ occurrences de E. Cette hypothèse était nécessaire dans le premier cas, mais pas dans le second.

Comme précédemment, la réciproque est similaire (enlever un E au lieu de l'ajouter), et on obtient $E^{k+1} \equiv_{[\![\phi]\!]} E^{k+2}$. En conclusion, $\mathbf{S}_{[\![\phi]\!]}$ est apériodique.

Cas $\phi = \varphi \mathbf{U}^{\leq N} \psi$

Ce cas ne présente pas de nouvelle difficulté importante par rapport au cas précédent. Il suffit de reprendre la preuve ci-dessus, en incorporant des erreurs possibles sur les positions où φ est vraie.

On s'aperçoit alors que supprimer une occurrence de E conserve le nombre d'erreurs, mais en ajouter une peut le doubler, puisque 2 positions de v_n sont parfois associées à la même position de u_n, comme on le voit sur la Figure 4.1. Cela se traduira par une borne M qui pourra doubler celle donnée par l'hypothèse, contrairement aux cas précédents. Comme on s'intéresse au caractère borné ou non de suites et non aux valeurs précises, ceci n'est pas génant.

En utilisant les hypothèses $E^k \equiv_{[\![\psi]\!]} E^{k+1}$ et $E^k \equiv_{[\![\varphi]\!]} E^{k+1}$, on obtient donc $E^{k+1} \equiv_{[\![\varphi \mathbf{U}^{\leq N} \psi]\!]} E^{k+2}$. On peut donc conclure que $\mathbf{S}_{[\![\phi]\!]}$ est aussi apériodique dans ce cas. □

4.4.2 De l'apériodicité à CLTL

On va montrer montrer la réciproque du Théorème 4.4.4, c'est-a-dire le théorème suivant :

Théorème 4.4.5 *Soit f une fonction de coût régulière telle que $\mathbf{S}_f$ est apériodique, alors il existe une formule φ de CLTL telle que $f \approx [\![\varphi]\!]$.*

Démonstration

Cette preuve généralise la preuve de Wilke pour les langages apériodiques donnée dans [Wil99]. Cependant, des difficultés inhérentes aux notions quantitatives apparaissent ici.

On commence par ajouter un élément neutre 1 à $\mathbf{S}_f$ s'il n'y en avait pas. Cela ne change pas son apériodicité, ni sa capacité à reconnaître f. Soit $\mathbf{M} = \langle M, \cdot, \leq, \sharp \rangle$ le monoïde de stabilisation obtenu.

Pour éviter l'utilisation d'une fonction $h : \mathbb{A} \to \mathbf{M}$ tout au long de la preuve, on va considérer (sans perte de généralité) que l'on travaille sur un alphabet $\mathbb{A} \subseteq \mathbf{M}$. Pour des raisons de minimalité, on pourra toujours supposer que $M = \langle \mathbb{A} \rangle^\sharp \cup \{1\}$, c'est-à-dire que l'on restreint M aux éléments utiles.

Les formules atomiques de CLTL sont donc des éléments de M. On va supposer également qu'elles sont monotones : chaque lettre $a \in M$ va en réalité accepter toute lettre $b \geq a$. Cela siginifie que $[\![a]\!](bu) = 0$ si $b \geq a$, et ∞ sinon. Il n'est pas difficile d'obtenir le cas général (avec $\mathbb{A}$ et M disjoints) à partir de ces formules, en substituant dans la formule CLTL obtenue chaque élément m par $\vee_{h(a) \geq m} a$.

Pour simplifier la preuve, on ne détaillera pas le cas du mot vide ε. Ce mot ne présente pas de difficulté particulière, mais le prendre en compte et faire un cas spécial à toutes les étapes alourdirait la preuve sans nécessité.

On considère donc f sur un alphabet $\mathbb{A} \subseteq M$, reconnue par $\mathbf{M}, I$ (h est devenu l'identité) avec $\mathbf{M}$ apériodique et $1 \in M$. Soit ρ compatible avec $\mathbf{M}$.

Pour tout $m \in M$, on note f_m la fonction de coût définie par

$$f_m(u) = \inf \{n \in \mathbb{N} : \rho(u)(n) \geq m\} ..$$

Il suffit de montrer que les f_m sont CLTL-définissable, puisque $f \approx \min_{m \notin I} f_m$.

On procède par induction sur $(|M|, |\mathbb{A}|)$, où les couples de $\mathbb{N}^2$ sont ordonnés par l'ordre lexicographique $\leq_{lex}$. On ajoute aussi dans l'hypothèse d'induction la présence d'un élément neutre 1 pour le produit dans $\mathbf{M}$.

Si $|\mathbf{M}| = 1$, alors f est une fonction constante égale à 0 ou ∞, et est donc CLTL-définissable.

Si $\mathbb{A} = \{a\}$, alors on montre que $M = \{a^i : 0 \leq i \leq p\} \cup \{(a^p)^\sharp, 1\}$. En effet, soit p l'indice d'idempotence de a, alors $\langle \mathbb{A} \rangle = \{a^i : 0 \leq i \leq p\}$. Le seul idempotent de cet ensemble est a^p, qui génère donc un nouvel idempotent $(a^p)^\sharp$ par stabilisation. On sait que $(a^p)^\sharp$ est stable, et de plus $(a^p)^\sharp a = (a^p)^\sharp a^p a = (a^p)^\sharp a^p = (a^p)^\sharp$, par apériodicité de $\mathbf{M}$. De même $a(a^p)^\sharp = (a^p)^\sharp$. Il n'y a donc plus de moyen de générer de nouvel élément, et $M = \{a^i : 0 \leq i \leq p\} \cup \{(a^p)^\sharp, 1\}$. De plus, $(a^p)^\sharp \leq a^p$ est la seule paire dans $\leq$. On peut alors montrer que pour tout $b \in \mathbf{M}$, f_b est CLTL-définissable :

- si $a \neq 1$ alors $f_1 = [\![\Omega]\!]$.
- Si $i < p$, $f_{a^i} \approx [\![\mathbf{X}^{i-1} a \wedge \mathbf{X}^i \Omega]\!]$,
- $f_{a^p} \approx [\![\bot \mathbf{U}^{\leq N} \Omega]\!]$,
- $f_{(a^p)^\sharp} \approx [\![\mathbf{X}^p a]\!]$

On suppose maintenant $|\mathbf{M}| > 1$, $|\mathbb{A}| > 1$, et le résultat est vrai pour $(|\mathbf{M}'|, |\mathbb{A}'|) <_{lex} (|\mathbf{M}|, |\mathbb{A}|)$. Soit $b \neq 1 \in \mathbb{A}$, et $\mathbb{B} = \mathbb{A} \setminus \{b\}$.

Soit $L_0 = \mathbb{B}^*$, $L_1 = \mathbb{B}^* b \mathbb{B}^*$, et $L_2 = \mathbb{B}^* b (\mathbb{B}^* b)^+ \mathbb{B}^*$. On a alors $\mathbb{A}^* = L_0 \cup L_1 \cup L_2$.

On définit les restrictions de f_m : f_0, f_1, f_2 sur L_0, L_1, L_2 respectivement (la valeur étant ∞ en dehors du domaine). On a $f_m = \min(f_0, f_1, f_2)$. Il suffit donc

de montrer que les f_i sont CLTL-définissables pour obtenir que f_m est aussi CLTL-définissable, puisqu'une disjonction de formules s'évalue en leur minimum.

Premièrement, f_0 est reconnue par $\mathbf{M}$ sur l'alphabet $\mathbb{B}$ de taille strictement inférieure à $|\mathbb{A}|$, donc par hypothèse d'induction, il existe une formule CLTL φ_0 sur $\mathbb{B}$ reconnaissant f_0. La formule $\varphi_0' = \varphi_0 \wedge \mathbf{G}\neg b$ est donc une formule sur $\mathbb{A}$ reconnaissant f_0.

On montre maintenant que f_1 est CLTL-définissable. Pour tout $x \in \mathbf{M}$, soit φ_x la formule CLTL sur $\mathbb{B}$ reconnaissant f_x (restreinte à $\mathbb{B}^*$). Ces formules existent par hypothèse d'induction car $|\mathbb{B}| < |\mathbb{A}|$.

Si φ est une formule CLTL sur $\mathbb{B}$, on définit sa « relativisation » φ' sur $\mathbb{A}$, qui mime l'effet de φ sur le mot tronqué par le premier b. On définit φ' par induction de la manière suivante :

$$\begin{array}{lcl} a' & = & a \wedge \mathbf{XF}b \\ \Omega' & = & b \\ (\varphi \wedge \psi)' & = & \varphi' \wedge \psi' \\ (\varphi \vee \psi)' & = & \varphi' \vee \psi' \\ (\mathbf{X}\varphi)' & = & \mathbf{X}\varphi' \wedge \neg b \\ (\varphi \mathbf{U} \psi)' & = & (\varphi' \wedge \neg b)\mathbf{U}\psi' \\ (\varphi \mathbf{U}^{\leq N} \psi)' & = & (\varphi' \mathbf{U}^{\leq N} \psi') \wedge (\neg b \mathbf{U} \psi') \end{array}$$

Avec cette définition, il est facile de voir que $[\![\varphi']\!](u_1 b u_2) = [\![\varphi]\!](u_1)$ pour tout $u_1 \in \mathbb{B}^*$ et $u_2 \in \mathbb{A}^*$.

On définit la formule suivante sur $\mathbb{A}$:

$$\varphi_1 = (\bigvee_{xby=m} (\varphi'_x \wedge \mathbf{F}(b \wedge \mathbf{X}\varphi_y)) \wedge (\neg b \mathbf{U}(b \wedge \mathbf{XG}\neg b))$$

La seconde partie contrôle que le mot donné en argument est dans L_1. On va montrer que $[\![\varphi_1]\!] \approx f_1$.

Soit $u \in L_1$, on peut écrire $u = u_1 b u_2$ avec $u_1, u_2 \in \mathbb{B}^*$.
Par définition de φ_1,

$$\begin{array}{ll} [\![\varphi_1]\!](u) & = \min_{xby=m} \max([\![\varphi'_x]\!](u), [\![\varphi_y]\!](u_2)) \\ & = \min_{xby=m} \max([\![\varphi_x]\!](u_1), [\![\varphi_y]\!](u_2)) \\ & = \min_{xby=m} \max(f_x(u_1), f_y(u_2)). \end{array}$$

Pour tout $z \in \mathbf{M}$ et $v \in \mathbb{B}^*$, par définition de f_z, $\rho(v) \succeq \perp|_{f_z(v)} z$ où $\perp$ est un absorbant, minimal pour $\leq$ (on peut l'ajouter au semigroupe si nécessaire)

Mais pour tous x, y tels que $xby = m$,

$$\begin{array}{ll} \rho(u) & \sim \tilde{\rho}(\rho(u_1) b \rho(u_2)) \\ & \succeq \tilde{\rho}(\perp|_{f_x(u_1)} x \cdot b \cdot \perp|_{f_y(u_2)} y) \\ & \succeq \perp|_{\max(f_x(u_1), f_y(u_2))} m. \end{array}$$

Il existe donc β (indépendant de u), tel que pour tous x, y avec $xby = m$, $f_m(u) \leq_\beta \max(f_x(u_1), f_y(u_2))$.

En particulier, $f_1(u) = f_m(u) \leq_\beta \min_{xby \in I} \max(f_x(u_1), f_y(u_2)) = [\![\varphi_1]\!](u)$. On peut conclure $f_1 \preccurlyeq [\![\varphi_1]\!]$.

Réciproquement, supposons $f_1(u) \leq n$, cela signifie que $\rho(u)(n) \geq m$. Mais $\rho(u) \sim_\alpha \rho(u_1) \cdot b \cdot \rho(u_2)$, donc $\rho(u_1)(\alpha(n)) \cdot b \cdot \rho(u_2)(\alpha(n)) \geq m$.

Soit $x = \rho(u_1)(\alpha(n))$ et $y = \rho(u_2)(\alpha(n))$, on a $f_x(u_1) \leq \alpha(n)$ et $f_y(u_2) \leq \alpha(n)$, donc $\max(f_x(u_1), f_y(u_2)) \leq \alpha(n)$. On obtient $[\![\varphi_1]\!](u) \leq \alpha(n)$, et en conclusion $[\![\varphi_1]\!] \preccurlyeq f_1$. Ceci conclut la preuve de $[\![\varphi_1]\!] \approx f_1$.

Il nous reste à montrer que f_2 est CLTL-définissable. Pour ceci, on va finalement utiliser l'hypothèse d'induction sur la taille de $\mathbf{M}$ (jusqu'à maintenant, seul l'alphabet diminuait).

On définit le semigroupe de stabilisation $\mathbf{M'} = \langle Mb \cap bM, \circ, \natural, \leq' \rangle$ de la manière suivante : $xb \circ by = xby$, et pour tout xb idempotent, $(xb)^\natural = (x^\omega)^\sharp b$, où x^ω est l'idempotent obtenu en itérant x. Puisque $\mathbf{M}$ est apériodique, on peut toujours prendre $x^\omega = x^{|\mathbf{M}|}$. On peut vérifier que $\mathbf{M'}$ est un semigroupe de stabilisation, soit ρ' compatible avec $\mathbf{M'}$.

On peut remarquer que d'après cette définition, pour tout $k \in \mathbb{N}$ et $xb \in \mathbf{M'}$, $(xb)^k = x^k b$, donc $\mathbf{M'}$ est aussi apériodique. De plus, on montre que l'élément 1, qui est dans $\mathbf{M}$ par hypothèse, n'est plus dans $\mathbf{M'}$: Supposons $1 \in \mathbf{M'}$, soit $k = |\mathbf{M}|$, alors $1 = xb = x1b = x(xb)b = \cdots = x^k b^k = x^k b^{k+1} = 1b = b$, mais $b \neq 1$, c'est absurde donc $1 \notin \mathbf{M'}$. On peut également remarquer que b est neutre pour $\circ$ dans $\mathbf{M'}$, et $|\mathbf{M'}| < |\mathbf{M}|$. On pourra donc utiliser l'hypothèse d'induction sur $\mathbf{M'}$ avec alphabet $\mathbf{M'}$.

Soit $\Delta = b(\mathbb{B}^* b)^+$, alors $L_2 = \mathbb{B}^* \Delta \mathbb{B}^*$.

Soit $d \in \mathbf{M'}$, on veut montrer que f_d restreinte au langage Δ est CLTL-définissable.

On rappelle que pour tout ensemble E, on identifie les éléments de $(E^{\mathbb{N}})^*$ (mot de suites) avec leur projection canonique dans $(E^*)^{\mathbb{N}}$ (suites de mots), qui est toujours une suite de mots de longueurs identiques.

Soit $\sigma : \Delta \to (\mathbf{M'}^{\mathbb{N}})^*$, définie par

$$\sigma(bu_1 b \dots u_k b) = (b\rho(u_1)b) \dots (b\rho(u_k)b).$$

Par hypothèse d'induction, pour tout $x \in \mathbf{M'}$, il existe une formule CLTL ψ_x sur l'alphabet $\mathbf{M'}$, et une fonction de correction α_x tels que pour tout $v \in \mathbf{M'}^*$,

$$[\![\psi_x]\!](v) \approx_{\alpha_x} \inf \{n \in \mathbb{N} : \rho'(v)(n) \geq x\}.$$

Définition 4.4.6 *Soit* $\mathbf{S}$ *un semigroupe de stabilisation et* α *une fonction de correction. soit* f *une fonction de coût* $\mathbf{S}^* \to \mathbb{N}^\infty$*, et* $\mathbf{S}^\uparrow$ *l'ensemble des suites* α*-croissantes d'éléments de* $\mathbf{S}$*. On définit* $\tilde{f} : (\mathbf{S}^\uparrow)^* \to \mathbb{N}_\infty$ *par* $\tilde{f}(\vec{u}) = \inf \{n : f(u_n) \leq n\}$*.*

On peut remarquer que cette notation est cohérente avec l'opérateur $\tilde{}$ défini dans la Section 2.2.5 pour les fonctions $S^+ \to \mathbb{N} \to S$, en vertu de la remarque suivante : si f est reconnue par $\mathbf{S}, h, I$ avec ρ compatible, c'est-à-dire $f \approx u \mapsto I[\rho(h(u))]$, alors $\tilde{f} \approx \vec{u} \mapsto I[\tilde{\rho}(h(\vec{u}))]$.

Cette définition est nécessaire car σ est définie à partir de ρ, qui associe à chaque mot de $\mathbb{B}^*$ une suite d'éléments. Cependant, on veut recombiner ces éléments pour voir leur comportement vis-à-vis de f, et on a donc besoin d'une fonction qui prend comme argument des suites, et non plus des mots. C'est le

rôle de l'opérateur $\tilde{}$: changer le type de la fonction f tout en conservant sa signification, au moyen d'un procédé diagonal.

Lemme 4.4.7 *Il existe α et ϕ_d une formule CLTL sur $\mathbb{A}$ tel que pour tout $u \in \Delta$ et $v \in \mathbb{B}^*$:*

$$[\![\phi_d]\!](uv) \approx_\alpha \widetilde{[\![\psi_d]\!]}(\sigma(u)) \approx_\alpha f_d(u)$$

Intuitivement, ϕ_d oublie la dernière composante v de $\mathbb{B}^*$ donnée en argument, et applique $\sigma(u)$ pour factoriser le mot en fonction des b, afin d'utiliser ψ_d pour lire chaque facteur comme une lettre unique.

En supposant ce lemme vrai, on peut construire une formule φ_2 pour f_2 :

$$\varphi_2 = (\bigvee_{xdy=m} (\varphi'_x \wedge F(b \wedge X\phi_d)) \wedge F(b \wedge X(G\neg b \wedge \varphi_y))) \wedge \varphi_{L_2}$$

où $\varphi_{L_2} = F(b \wedge XFb)$ contrôle que le mot est dans L_2.

Par construction, hypothèse d'induction, et Lemme 4.4.7, il existe α tel que pour tous $v_1, v_2 \in \mathbb{B}^*$ et $u \in \Delta$,

$$\begin{aligned}[\![\varphi_2]\!](v_1uv_2) &\approx_\alpha \min_{xdy=m} \max([\![\varphi'_x]\!](v_1uv_2), [\![\phi_d]\!](uv_2), [\![\varphi_y]\!](v_2)) \\ &\approx_\alpha \min_{xdy=m} \max(f_x(v_1), f_d(u), f_y(v_2)).\end{aligned}$$

La preuve que $min_{xdy=m} \max(f_x(v_1), f_d(u), f_y(v_2)) \approx f_m(v_1uv_2)$ est similaire à celle du cas $[\![\varphi_1]\!] \approx f_1$.

On a donc obtenu $[\![\varphi_2]\!] \approx f_2$, ce qui conclut la preuve.

□

Preuve du Lemme 4.4.7

Démonstration On commence par montrer $\widetilde{[\![\psi_d]\!]}(\sigma(u)) \approx_\alpha f_d(u)$ pour un certain α et tout $u \in \Delta$. Soit $u = bu_1bu_2 \dots u_kb$ avec $u_i \in \mathbb{B}^*$. Pour tout $i \in [\![1, k]\!]$ et $t \in \mathbb{N}$, $\rho(u_i)(t) = a_{i,t} \in \mathbf{M}$. Pour tout $t \in \mathbb{N}$, soit $v_t = (ba_{1,t}b) \dots (ba_{k,t}b)$, v_t est un mot sur $\mathbf{M}'$ de longueur k, et $\sigma(u) = (v_t)_{t\in\mathbb{N}}$. Finalement, soit $w_t = ba_{1,t}ba_{2,t} \dots ba_{k,t}b$ de longueur $2k+1$ sur $\mathbf{M}$. On a :

$$\begin{aligned}\widetilde{[\![\psi_d]\!]}(\sigma(u))) &= \inf\{t : [\![\psi_d]\!](v_t) \leq t\} \\ &\approx \inf\{t : \inf\{n : \rho'(v_t)(n) \geq d\} \leq t\}\end{aligned}$$

On peut montrer que $\rho'(v_t) \sim \rho(w_t)$ pour tout t. Il suffit de vérifier que ρ' comme ρ vérifient tous les axiomes du Théorème 2.2.14 sur $(ba_1b) \dots (ba_kb)$ et $ba_1ba_2 \dots a_kb$ respectivement. Soit α et α' les fonctions de corrections données par ce théorème pour ρ et ρ'.

Lettre. pour tout $a \in M$, on a bien $\rho'(bab) \sim_{\alpha'} bab \sim_\alpha \rho(bab)$,

Produit. pour tout $a_1, a_2 \in M$, on a $\rho'((ba_1b)(ba_2b)) \sim_{\alpha'} (ba_1b) \circ (ba_2b) = ba_1ba_2b \sim_{\alpha^4} \rho(ba_1ba_2b)$,

Stabilisation. Soit bab un idempotent de $\mathbf{M}'$ et $m \in \mathbb{N}$, Notons que $babab = bab$, donc pour tout $l \geq 1$, $(ba)^lb = bab$, où les produits sont effectués dans $\mathbf{M}$.

$\rho'((bab)^m) \sim_{\alpha'} (bab)^\natural|_m(bab) = (ba)^{\omega\sharp}b|_m(bab)$. On effectue la division euclidienne $m = |\mathbf{M}|m' + m''$ avec $m'' < |\mathbf{M}|$,

$$\begin{aligned}\rho((ba)^m b) &\sim \rho(((ba)^{|\mathbf{M}|})^{m'}) \cdot (ba)^{m''}b \\ &\sim^{(1)} \rho(((ba)^\omega)^{m'}) \cdot (bab) \\ &\sim (ba)^{\omega\sharp}(bab)|_{m'}(ba)^\omega(bab) \\ &\sim^{(2)} (ba)^{\omega\sharp}b|_m(bab).\end{aligned}$$

L'équivalence (1) est obtenue par apériodicité de $\mathbf{M}$ ($(ba)^\omega$ est une lettre et non un mot de longueur $|\mathbf{M}|$), et (2) en utilisant $m \approx_{\times(|\mathbf{M}|+1)} m'$.

Ces cas montrent qu'un n-calcul dans $\mathbf{M}'$ sur v_t peut être transformé en un n-calcul sur w_t de même valeur, car chaque type de noeud préserve cette propriété. La propriété de substitution correspond au branchement en cascade de plusieurs n-calculs, il n'est donc pas nécessaire de traiter ce cas ici.

On obtient donc $\rho'(v_t) \sim \rho(w_t)$ pour tout t.

De plus, soit $\vec{w} = (w_t)_{t\in\mathbb{N}}$, on montre que

$$\inf\{n' : \tilde{\rho}(\vec{w})(n') \geq d\} \approx \inf\{t : \inf\{n : \rho(w_t)(n) \geq d\} \leq t\} : (EQ).$$

Soit $N' = \inf\{n' : \tilde{\rho}(\vec{w})(n') \geq d\}, \rho(w_{N'})(N') \geq d$ et $N' \leq N'$
on a $N' \geq \inf\{t : \inf\{n : \rho(w_t)(n) \geq d\} \leq t\}$.

Réciproquement, soit $T = \inf\{t : \inf\{n : \rho(w_t)(n) \geq d\} \leq t\}$ et N la valeur correspondante de $\inf\{n : \rho(w_t)(n) \geq d\}$, on a $N \leq T$ et $\rho(w_t)$ est α-croissante, donc $\rho(w_T)(T) \geq_\alpha \rho(w_T)(N) \geq d$, c'est-à-dire $T \geq_\alpha \inf\{n' : \tilde{\rho}(\vec{w})(n') \geq d\}$.

On obtient donc l'équivalence (EQ).

Finalement,

$$\begin{aligned}\widetilde{[\![\psi_d]\!]}(\sigma(u))) &\approx \inf\{t : \inf\{n : \rho(w_t)(n) \geq d\} \leq t\} \\ &\approx \inf\{n : \tilde{\rho}(\vec{w})(n) \geq d\} && \text{par } (EQ) \\ &= \inf\{n : \tilde{\rho}(b\rho(u_1)b\rho(u_2)\dots\rho(u_k)b)(n) \geq d\} \\ &\approx \inf\{n : \rho(bu_1bu_2\dots u_kb)(n) \geq d\} && \text{Substitution} \\ &\approx f_d(u).\end{aligned}$$

ce qui conclut la preuve de $\widetilde{[\![\psi_d]\!]}(\sigma(u))) \approx f_d(u)$.

Il reste à montrer l'existence de ϕ_d et α tels que pour tous $u, v \in \Delta \times \mathbb{B}^*$, $[\![\phi_d]\!](uv) \approx_\alpha \widetilde{[\![\psi_d]\!]}(\sigma(u))$.

Si ψ est une formule sur $\mathbf{M}'$, on définit $\psi^\bigstar$ sur $\mathbb{A}$ par induction sur ψ :

$$\begin{aligned}x^\bigstar &= (b \wedge \mathbf{XF}b) \wedge (\mathbf{X}\varphi'_x) \\ (\psi_1 \wedge \psi_2)^\bigstar &= \psi_1^\bigstar \wedge \psi_2^\bigstar \\ (\psi_1 \vee \psi_2)^\bigstar &= \psi_1^\bigstar \vee \psi_2^\bigstar \\ (\mathbf{X}\psi)^\bigstar &= \neg b\mathbf{U}(b \wedge \psi^\bigstar) \\ (\psi_1\mathbf{U}\psi_2)^\bigstar &= (b \implies \psi_1^\bigstar)\mathbf{U}(b \wedge \psi_2^\bigstar) \\ (\psi_1\mathbf{U}^{\leq N}\psi_2)^\bigstar &= (b \implies \psi_1^\bigstar)\mathbf{U}^{\leq N}(b \wedge \psi_2^\bigstar).\end{aligned}$$

Où φ'_x est définie comme précédemment, pour tout φ_x sur l'alphabet $\mathbb{B}$.

On peut maintenant montrer par induction sur ψ que $[\![\psi^\bigstar]\!](uv) \approx \widetilde{[\![\psi]\!]}(\sigma(u))$ pour $u = bu_1bu_2\dots u_kb \in \Delta$ et $v \in \mathbb{B}^*$:

– Si $x \in \mathbf{M}'$,
$[\![x^{\bigstar}]\!](uv) = [\![\varphi'_x]\!](u_1 b u_2 \dots u_k b v) = [\![\varphi_x]\!](u_1)$, et
$\widetilde{[\![x]\!]}(\sigma(u)) = \inf\{n : [\![x]\!](\rho(u_1)(n)) \leq n\}$
$\approx \inf\{n : (\rho(u_1)(n)) \geq x\}$
$\approx [\![\varphi_x]\!](u_1)$.

– Cas $\wedge$:
$[\![(\psi_1 \wedge \psi_2)^{\bigstar}]\!](uv) = \max([\![\psi_1^{\bigstar}]\!](uv), [\![\psi_2^{\bigstar}]\!](uv))$
$\approx \max(\widetilde{[\![\psi_1]\!]}(\sigma(u)), \widetilde{[\![\psi_2]\!]}(\sigma(u)))$
$\approx \widetilde{[\![\psi_1 \wedge \psi_2]\!]}(\sigma(u))$

– Cas $\vee$:
$[\![(\psi_1 \vee \psi_2)^{\bigstar}]\!](uv) = \min([\![\psi_1^{\bigstar}]\!](uv), [\![\psi_2^{\bigstar}]\!](uv))$
$\approx \min(\widetilde{[\![\psi_1]\!]}(\sigma(u)), \widetilde{[\![\psi_2]\!]}(\sigma(u)))$
$\approx \widetilde{[\![\psi_1 \vee \psi_2]\!]}(\sigma(u))$

– cas $\mathbf{X}$:
$[\![(X\psi)^{\bigstar}]\!](uv) = \widetilde{[\![\psi^{\bigstar}]\!]}(bu_2 b \dots u_k b v)$
$\approx \widetilde{[\![\psi]\!]}(\sigma(bu_2 b \dots u_k b))$
$\approx \widetilde{[\![X\psi]\!]}(\sigma(bu_1 bu_2 b \dots u_k b))$

– cas $\mathbf{U}$:
$[\![(\psi_1 U \psi_2)^{\bigstar}]\!](uv)$
$= \min_{1 \leq j \leq k}(\max([\![\psi_2^{\bigstar}]\!](bu_j b \dots u_k b v), \max_{1 \leq i \leq j} [\![\psi_1^{\bigstar}]\!](bu_i b \dots u_k b v)))$
$\approx \min_{1 \leq j \leq k}(\max(\widetilde{[\![\psi_2]\!]}(\sigma(bu_j b \dots u_k b)), \max_{1 \leq i \leq j} \widetilde{[\![\psi_1]\!]}(\sigma(bu_i b \dots u_k b))))$
$\approx \widetilde{[\![\psi_1 U \psi_2]\!]}(\sigma(u))$

– Le cas $\mathbf{U}^{\leq N}$ est le même que ci-dessus, en autorisant au plus N erreurs pour ψ_1.
Il suffit maintenant de choisir $\phi_d = \psi_d^{\bigstar}$ pour compléter la preuve du Lemme 4.4.7.

□

Ceci conclut la preuve du Théorème 4.4.3

4.4.3 Décidabilité

La logique CLTL est une manière très concise (comparée par exemple aux automates de coût et aux semigroupes de stabilisation) de représenter des fonctions de coût, par exemple pour décrire un comportement souhaité pour un système. Il est donc naturel, étant donnée une fonction de coût, de se demander s'il existe une formule de CLTL pour la décrire.

Le théorème suivant donne un résultat positif dans ce sens :

Théorème 4.4.8 *L'appartenance à la classe des fonctions définissables par CLTL est décidable pour les fonctions de coût regulières.*

Démonstration Ce théorème découle de la caractérisation donnée par le Théorème 4.4.3.

En effet, si f est donnée par un automate ou un semigroupe de stabilisation, on peut calculer $\mathbf{S}_f$ (voir Section 2.3.6). Il suffit ensuite de tester l'apériodicité de $\mathbf{S}_f$ pour répondre à la question. Ceci peut être fait en temps quadratique en $n = |\mathbf{S}_f|$, en itérant au plus n fois chaque élément $a \in \mathbf{S}_f$ jusqu'à détecter un cycle. Si tous les cycles sont triviaux, alors le semigroupe est apériodique, sinon il ne l'est pas. Attention, si f est donnée par un automate, cette procédure devient exponentielle en la taille de l'entrée, car le passage de l'automate au semigroupe de stabilisation est exponentiel, voir [Col11]. □

4.5 Etude de Prompt-LTL

4.5.1 Définition, lien avec les fonctions de coût

La logique Prompt-LTL a été introduite dans [KPV09], afin d'enrichir LTL par une contrainte quantitative. Plus précisément, cela se fait au moyen d'un nouvel opérateur $\mathbf{F_P}$. La syntaxe de Prompt-LTL est donc

$$\varphi := a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid \mathbf{F_P}\varphi \mid \Omega.$$

La syntaxe est ici légèrement différente de l'originale car on se restreint aux mots finis. Ce travail pourra ensuite être étendu aux mots infinis.

Le nouvel opérateur $\mathbf{F_P}$ se comporte comme $\mathbf{F}$, mais nécessite une borne globale sur le temps d'attente. Formellement, si φ est une formule de Prompt-LTL u est un mot et $k \in \mathbb{N}$, on dit que $(u, k) \models \varphi$ si $u \models \varphi_k$ où φ_k est obtenue à partir de φ en remplaçant chaque occurence de $\mathbf{F_P}$ par $\bigvee_{i \leq k} \mathbf{X}^i$.

Un ensemble de mots E (défini par un système de transitions fini dans [KPV09]) satisfait φ, noté $E \models \varphi$, s'il existe une borne k tel que pour tout mot $u \in E$, $(u, k) \models \varphi$.

Bien que cette logique ait été introduite indépendemment de la théorie des fonctions de coût, on peut voir Prompt-LTL comme un moyen de définir des fonctions de coût. Ce fait est explicité par le Lemme suivant.

Lemme 4.5.1 *Soit φ est une formule de Prompt-LTL, et φ' la formule de CLTL obtenue à partir de φ en remplaçant chaque $\mathbf{F_P}\psi$ par $\bot \mathbf{U}^{\leq N} \psi$. Alors pour tout ensemble de mots E, $E \models \varphi$ si et seulement si $[\![\varphi']\!]$ est borné sur E.*

Démonstration La définition de $E \models \varphi$ correspond à l'existence d'une borne globale k sur le temps d'attente utilisable pour tout mot $u \in E$. C'est équivalent à l'existence d'une borne pour $[\![\varphi']\!]$ sur E, par définition de φ'. □

Dans la suite on pourra donc utiliser l'opérateur $\mathbf{F_P}\psi$ comme une notation pour $\bot \mathbf{U}^{\leq N} \psi$, et considérer les formules de Prompt-LTL comme des formules de CLTL.

4.5.2 Pouvoir expressif de Prompt-LTL

On peut maintenant se demander quel est le pouvoir expressif de Prompt-LTL en terme de fonctions de coût reconnues. Chaque formule de Prompt-LTL étant en particulier une formule de CLTL, on sait déjà par le Théorème 4.4.3 que toute fonction reconnue par une formule de Prompt-LTL est apériodique (c'est-à-dire que son semigroupe de stabilisation minimal est apériodique).

On cherche une caractérisation algébrique de la classe des fonctions de coût reconnues par Prompt-LTL, et si possible un résultat sur la décidabilité de cette classe. Intuitivement, les évènements mesurés par une formule de Prompt-LTL sont toujours consécutifs, et on s'attend donc à ce qu'une fonction de coût définie ainsi soit temporelle.

Lemme 4.5.2 *Soit ϕ une formule de Prompt-LTL. Alors $[\![\phi]\!]$ est temporelle.*

Démonstration On va montrer qu'il est possible de construire un automate temporel reconnaissant $[\![\phi]\!]$. La construction donnée dans la Section 4.2, donnant un B-automate $\mathcal{A}_\phi$ pour toute formule ϕ de CLTL, ne fonctionne pas directement.

Il y a plusieurs raisons à cela :

- Chaque opérateur classique ($\wedge, \vee, \mathbf{U}$) génère des actions ε dans l'automate.
- L'automate obtenu possède plusieurs compteurs, et les actions générées n'agissent que sur un compteur à la fois, effectuant l'action ε sur les autres.

On doit donc modifier cet automate de façon à le rendre temporel. On numérote les occurrences de $\mathbf{F_P}$ en $\mathbf{F_P}_1, \mathbf{F_P}_2, \ldots, \mathbf{F_P}_k$ dans la formule ϕ. On veut construire $\mathcal{A}_\phi = \langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta \rangle$ temporel reconnaissant $[\![\phi]\!]$.

Comme précédemment, on pose $Q = 2^{\mathrm{sub}}\phi$, $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$, $In = \{\{\phi\}\}$ et $Fin = \{\emptyset, \{\Omega\}\}$.

Le schéma général de définition des transitions est défini de la même manière que dans la section 4.2, avec des ε-transitions reliant des pseudo-états.

La seule différence se situe dans la définition suivante :

Soit $Y \in Q$, et ψ une formule non réduite de taille maximale dans Y. On ajoute les ε-transitions suivantes :

- Si $\psi = \varphi_1 \wedge \varphi_2$: $Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, \varphi_2\}$
- Si $\psi = \varphi_1 \vee \varphi_2$: $\left\{ \begin{array}{l} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{array} \right.$
- Si $\psi = \varphi_1 \mathbf{U} \varphi_2$: $\left\{ \begin{array}{l} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, \mathbf{X}\psi\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{array} \right.$
- Si $\psi = \mathbf{F_P}_j \varphi$: $\left\{ \begin{array}{l} Y \xrightarrow{\varepsilon:\mathtt{ic}_j} Y \setminus \{\psi\} \cup \{\mathbf{X}\psi\} \text{ (on compte une erreur)} \\ Y \xrightarrow{\varepsilon:\mathtt{r}_j} Y \setminus \{\psi\} \cup \{\varphi\} \end{array} \right.$

  où l'action $\mathtt{r}_j$ (resp. $\mathtt{ic}_j$) effectue l'action $\mathtt{r}$ (resp. $\mathtt{ic}$) sur le compteur γ_j et ε sur les autres compteurs.

Les transitions réelles sont comme auparavant obtenues par contraction des ε-transitions. De plus, pour chaque compteur si l'action globale est ε le long d'une séquence d'ε-transitions, on effectue une action $\mathtt{r}$ sur ce compteur dans

la transition réelle, afin d'obtenir un automate temporel. Les transitions de l'automate $\mathcal{A}_\phi$ sont donc définies comme suit :

$$\Delta = \left\{ Y \xrightarrow{a:\sigma'} \text{next}(Z) \mid Y \in Q, Z \cup \{a\} \text{ consistant et réduit}, Y \xrightarrow{\varepsilon:\sigma}_* Z \right\}$$

où pour tout $\gamma \in \Gamma$, $\sigma'(\gamma) = \begin{cases} \mathtt{r} \text{ si } \sigma(\gamma) = \varepsilon \\ \sigma(\gamma) \text{ sinon} \end{cases}$

La correction de cet automate est déduite du fait que si l'on veut effectuer l'action ε sur le compteur γ_j correspondant à l'opérateur $\mathbf{F}_{\mathbf{P}\,j}$, cela signifie que l'on n'est pas en train de compter le temps d'attente lié à $\mathbf{F}_{\mathbf{P}\,j}$. On peut donc remplacer ε par $\mathtt{r}$ sans danger, et obtenir ainsi un automate $\mathcal{A}_\phi$ temporel pour ϕ. □

La classe des fonctions reconnues par Prompt-LTL est donc incluse dans l'intersection entre la classe temporelle et la classe apériodique. Le théorème suivant exprime la réciproque.

Théorème 4.5.3 *Soit f une fonction de coût temporelle et apériodique. Alors il existe une formule φ de Prompt-LTL telle que $f \approx [\![\varphi]\!]$.*

Démonstration On reprend la preuve du Théorème 4.4.5 de la Section 4.4.2, qui permettait de passer d'un semigroupe apériodique à une formule de CLTL. On suppose cette fois que le semigroupe $\mathbf{M}$ de départ est apériodique et temporel, on cherche à obtenir une formule ϕ_x de Prompt-LTL, pour chaque élément $x \in \mathbf{M}$. Cependant, on imposait dans l'hypothèse de récurrence que $1 \in \mathbf{M}$. Ceci est incompatible avec l'hypothèse selon laquelle $\mathbf{M}$ est temporel (dans le cas où il y a des éléments instables), on supprime donc cette hypothèse.

Le schéma de preuve reste exactement le même : on procède par induction sur $(|\mathbf{M}|, |\mathbb{A}|)$. On ne décrira ici que les nouveaux éléments de la preuve.

Les cas $|\mathbf{M}| = 1$ et $|\mathbb{A}| = 1$ n'utilisaient déjà que des formules de Prompt-LTL, et n'utilisaient pas l'hypothèse $1 \in \mathbf{M}$, on peut donc réutiliser cette partie de la preuve telle quelle. Remarquons également que si tous les éléments de $\mathbf{M}$ sont stables, alors les théorèmes sur les langages classiques (Schützenberger, Kamp) nous donnent directement une formule de LTL pour chaque élément.

On suppose maintenant $|\mathbf{M}| > 1$, $|\mathbb{A}| > 1$, et également qu'il existe un élément instable, le résultat étant vrai dans les autres cas.

Soit $b \in \mathbb{A}$ un élément stable, et $\mathbb{B} = \mathbb{A} \setminus \{b\}$. Un tel élément b existe toujours : on peut choisir par exemple $(x^\omega)^\sharp$ pour n'importe quel élément x.

Soit $L_0 = \mathbb{B}^*$, $L_1 = \mathbb{B}^* b \mathbb{B}^*$, et $L_2 = \mathbb{B}^* b (\mathbb{B}^* b)^+ \mathbb{B}^*$. On a alors $\mathbb{A}^* = L_0 \cup L_1 \cup L_2$.

Si φ est une formule CLTL sur $\mathbb{B}$, on définit sa « relativisation » φ' sur $\mathbb{A}$, qui mime l'effet de φ sur le mot tronqué par le premier b. On définit φ' par

induction de la manière suivante :

$$\begin{array}{lcl} a' & = & a \wedge \mathbf{XF}b \\ \Omega' & = & b \\ (\varphi \wedge \psi)' & = & \varphi' \wedge \psi' \\ (\varphi \vee \psi)' & = & \varphi' \vee \psi' \\ (\mathbf{X}\varphi)' & = & \mathbf{X}\varphi' \wedge \neg b \\ (\varphi \mathbf{U} \psi)' & = & (\varphi' \wedge \neg b)\mathbf{U}\psi' \\ (\mathbf{F_P}\psi)' & = & (\mathbf{F_P}\psi') \wedge (\neg b \mathbf{U} \psi') \end{array}$$

Le reste des cas L_0 et L_1 n'utilise pas l'opérateur $\mathbf{U}^{\leq N}$, donc on reste dans le fragment Prompt-LTL. Il reste à traiter le cas L_2.

La définition du semigroupe de stabilisation $\mathbf{M}'$ sera légèrement différente ici. En effet, on avait défini auparavant ses éléments $M' = Mb \cap bM$, afin que b soit le nouvel élément neutre, et que 1 disparaisse. Hormis cette considération, les seuls éléments utiles de $\mathbf{M}'$ étaient de la forme bab.

On définit donc ici $\mathbf{M}' = \langle bMb, \circ, \natural, \leq' \rangle$, avec $bxb \circ byb = bxbyb$, et pour tout bxb idempotent, $(bxb)^\natural = ((bx)^\omega)^\sharp b$, où $(bx)^\omega$ est l'idempotent obtenu en itérant bx. Ces définitions sont cohérentes avec la preuve précédente, on se restreint juste aux éléments utiles.

Or, cette fois-ci, on sait que b est stable, et $\mathbf{M}$ est temporel donc tout élément $\mathcal{J}$-inférieur à b est stable. On a donc pour tout bxb idempotent, $(bxb)^\natural = ((bx)^\omega)^\sharp b = (bx)^\omega b = bxb$. La dernière égalité vient du fait que $bxbxb = bxb$, et donc pour tout $l \geq 1$, $(bx)^l b = bxb$.

Le semigroupe de stabilisation $\mathbf{M}'$ défini ici est donc entièrement stable, c'est un semigroupe classique. La logique LTL classique est donc suffisante pour définir toutes les fonctions de $\mathbf{M}'$. Lorsque que l'on définit l'opérateur $\bigstar$, il n'est donc pas nécessaire de considérer le cas $\psi_1 \mathbf{U}^{\leq N} \psi_2$ comme dans la preuve précédente.

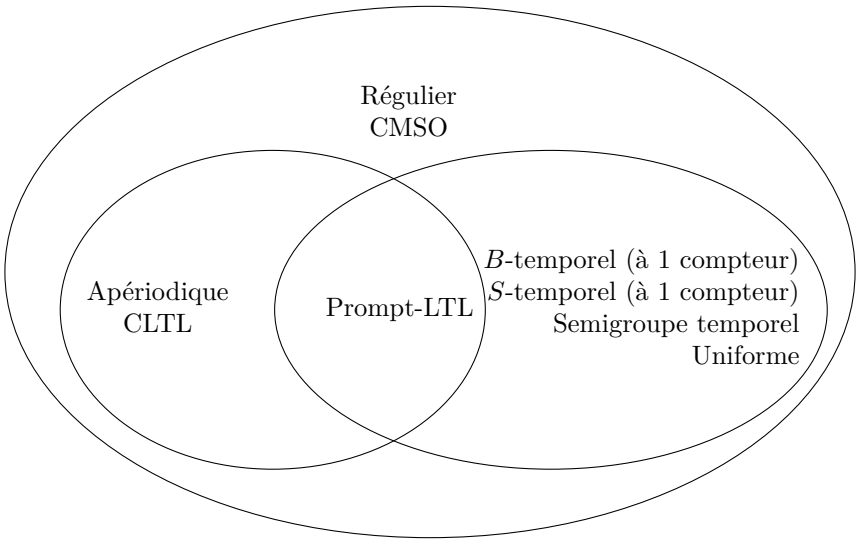
En réutilisant le reste de la preuve tel quel, on construit une formule de Prompt-LTL pour toute fonction f reconnue par $\mathbf{M}$. □

Corollaire 4.5.4 *Etant donnée une fonction de coût régulière, on peut décider si elle peut être reconnue par une formule de Prompt-LTL, en vérifiant si le semigroupe de stabilisation minimal de f est apériodique et temporel.*

Chapitre 5

Conclusion

On a étudié dans cette partie plusieurs classes de fonctions de coût sur mots finis, et caractérisé finement leur structure au moyen de plusieurs formalismes équivalents. Pour toutes ces classes, on a obtenu la décidabilité du problème de l'appartenance, au moyen d'une caractérisation algébrique à la manière de Schützenberger. La figure suivante résume la manière dont ces classes intéragissent. Les traductions d'un formalisme à l'autre en suivant ce schéma sont toutes effectives.



L'un des outils centraux mis en oeuvre ici est la notion d'$\omega\sharp$-expression, qui permet de définir une congruence syntactique sur les fonctions de coût. Ceci nous autorise à décrire le comportement du semigroupe minimal de f lorsque l'on a pas directement accès à celui-ci, comme c'est le cas par exemple si f est décrit par une formule de CLTL.On a de plus obtenu un résultat encourageant de complexité : le problème de l'existence de borne pour une formule de CLTL

est PSPACE-complet, tout comme le problème de satisfaisabilité de LTL qu'il généralise. En étudiant la classe temporelle , on a exhibé un lien explicite avec la théorie des langages via les langages uniformes, ce qui permet de simplifier grandement les structures manipulées ainsi que les approximations commises par les fonctions de correction.

Deuxième partie

Fonctions de coût régulières sur mots infinis

Chapitre 6

Formalismes de reconnaissance

On veut maintenant étendre la théorie des fonctions de coût régulières aux mots infinis.

La théorie des langages réguliers sur mots infinis a rencontrés de nombreux succès, et est aujourd'hui utilisée massivement en pratique, notamment dans un contexte de vérification. Le formalisme des mots infinis se justifie par le fait que beaucoup de systèmes informatiques réels sont utilisés en continu, et leur fin n'est pas programmée. On peut par exemple penser à l'exemple classique de l'ascenseur : si l'on veut modéliser son comportement de manière simple, cela a plus de sens de le considérer comme une système réagissant à des actions sur une durée infinie. On pourra ainsi spécifier que certaines conditions seront toujours vérifiées (par exemple à chaque fois qu'on l'appelle, l'ascenseur finira par venir), ou à l'inverse que certains comportements ne seront jamais possibles (par exemple rester bloqué entre deux étages).

On fixe comme précédemment un alphabet fini $\mathbb{A}$, et on note $\mathbb{A}^\omega$ l'ensemble des mots infinis sur $\mathbb{A}$. On peut supposer dans cette partie que $\mathbb{A}$ possède au moins deux lettres, sinon il n'existe qu'un seul mot et la théorie n'est pas très intéressante.

Une fonction de coût sur mot infinis est donc une classe d'équivalence de fonctions $\mathbb{A}^\omega \to \mathbb{N}_\infty$, pour la relation $\approx$ définie dans le Chapitre 1.

6.1 Automates de coût

On veut maintenant définir un modèle d'automates qui associeront un élément de $\mathbb{N}_\infty$ à chaque mot infini lu en entrée. On va étendre les automates de coût sur mots finis, en s'inspirant de la théorie des automates classiques sur mots infinis.

Les automates de Büchi, introduit dans [Büc62] forment la pierre angulaire de la théorie des langages réguliers sur mots infinis. Le principe de ces automates est le suivant : puisqu'il n'est plus possible d'examiner l'état final d'une exécution

(qui est maintenant de longueur infinie), la condition d'acceptation va être basée sur les états visités infiniment souvent. Plus précisément, un ensemble d'« états de Büchi » est choisi dans l'automate, et une exécution est acceptante si au moins l'un de ces états de Büchi a été visité infiniment souvent. Büchi a montré que ces automates généralisaient de manière satisfaisante les automates sur mots finis, en établissant leur équivalence avec la logique MSO sur mots infinis.

On va ici réutiliser cette approche, mais cette fois-ci avec des automates de coût. Il faut donc commencer par spécifier quelle valeur associer à une séquence infinie d'actions sur les B-compteurs et les S-compteurs.

6.1.1 B- et S-valuations

On garde les deux alphabets d'actions atomiques $\mathbb{C}_B = \{\mathtt{ic}, \varepsilon, \mathtt{r}\}$ et $\mathbb{C}_S = \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}$ pour les compteurs.

Cependant, on va maintenant vouloir assigner une valeur à un mot infini de telles actions. Les définitions $val_B(\nu) = \sup C(\nu)$ et $val_S(\nu) = \inf C(\nu)$ de la Section 2.1.1 seront conservées, mais pourront être étendues à $\nu \in (\mathbb{C}_B^\Gamma)^\omega$ dans et $\nu \in (\mathbb{C}_S^\Gamma)^\omega$ respectivement.

Ainsi on aura $val_B((\mathtt{ic}.\varepsilon.\mathtt{ic}.\mathtt{r})^\omega) = 2$ et $val_B(\mathtt{ic}.\mathtt{r}.\mathtt{ic}^2.\mathtt{r}.\mathtt{ic}^3.\mathtt{r}\dots) = \infty$, tandis que $val_S(\mathtt{i}.\mathtt{cr}.\mathtt{i}^2.\mathtt{cr}.\mathtt{i}^3.\mathtt{cr}\dots) = 1$ et $val_S(\mathtt{r}^\omega) = \infty$.

6.1.2 Automates de coût sur mots infinis

On va donner ici directement la définition de la classe générale des B- (resp. S-) automates de Büchi alternants sur mots infinis. Les diverses classes de B-automates sur mots infinis seront ensuite définies par des restrictions sur cette classe générale.

Un *B-automate de Büchi alternant* (B-ABA) $\mathcal{A} = \langle Q, \mathbb{A}, In, F, \Gamma, \delta\rangle$ sur mots infinis possède

- un ensemble fini d'états Q,
- un alphabet fini $\mathbb{A}$,
- un ensemble d'états initiaux $In \subseteq Q$,
- un ensemble d'états de Büchi $F \subseteq Q$,
- un ensemble fini Γ de B-compteurs. Soit $\mathbb{C} := \mathbb{C}_B^\Gamma$ l'ensemble des B-actions atomiques sur Γ.
- une fonction de transition $\delta : Q \times \mathbb{A} \to \mathcal{B}^+(\mathbb{C} \times Q)$, où $\mathcal{B}^+(\mathbb{C} \times Q)$ est l'ensemble des combinaisons booléennes positives (sans négation) d'éléments de $(\mathbb{C} \times Q)$. Pour tout $(q, a) \in Q \times \mathbb{A}$, on peut écrire $\delta(q, a)$ comme une disjonction de clauses, chaque clause étant une conjonction d'éléments $(c_i, q_i) \in \mathbb{C} \times Q$.

Le comportement d'un tel automate peut être vu comme un jeu entre deux joueurs : Eve et Adam. A chaque position q dans l'automate, si a est la lettre lue, Eve choisit une clause dans $\delta(q, a)$, puis Adam choisit un élément (c, q) dans cette clause, qui déterminera l'action effectuée et le nouvel état de l'automate.

Le but d'Eve est d'accepter le mot d'entrée avec la plus petite valuation possible pour l'action construite tout en validant la condition d'acceptation de

l'automate, et celui d'Adam est de l'en empêcher.

On appellera *stratégie* de $\mathcal{A}$ une stratégie d'Eve dans ce jeu, définie pour tous les choix possibles d'Adam.

Plus formellement, une stratégie de $\mathcal{A}$ sur $u = a_0 a_1 \cdots \in \mathbb{A}^\omega$ est un arbre infini étiqueté $S = (r, c)$ avec $dom(S) \subseteq \mathbb{N}^*$ (mots sur $\mathbb{N}$), $r : dom(S) \to Q$ et $c : (dom(S) \setminus \{\epsilon\}) \to \mathbb{C}$ vérifiant

- $r(\epsilon) \in In$;
- pour tout $x \in dom(S)$, il existe une clause $(c_1, q_1) \wedge \ldots \wedge (c_k, q_k)$ dans $\delta(r(x), a_{|x|})$ telle que pour tout $1 \leq j \leq k$, $x \cdot j \in dom(S)$, $r(x \cdot j) = q_j$, $c(x \cdot j) = c_j$, et pour tout $j > k$, $x \cdot j \notin dom(S)$.

On dit qu'une stratégie est acceptante si toute branche π de $S = (r, c)$ contient une infinité de noeuds x tels que $r(x) \in F$.

Une *exécution* de $\mathcal{A}$ est une branche d'une stratégie de $\mathcal{A}$: c'est un mot infini sur $Q \times \mathbb{C}$, fixée par les choix des deux joueurs sur le mot u donnée en entrée.

On définit maintenant la valeur d'une stratégie S de $\mathcal{A}$. Si $\pi = x_0 x_1 \ldots$ est une branche de S, la valeur de π est donnée par $val_B(\pi) := val_B(c(x_1)c(x_2)\ldots)$. On pose ensuite

$$val_B(S) = \sup \{ val_B(\pi) : \pi \text{ branche de } S \} .$$

La *B-sémantique* d'un B-ABA $\mathcal{A}$ est $[\![\mathcal{A}]\!]_B : \mathbb{A}^\omega \to \mathbb{N}_\infty$ définie par

$$[\![\mathcal{A}]\!]_B(u) := \inf \{ val_B(S) : S \text{ est une stratégie acceptante de } \mathcal{A} \text{ sur } u \} .$$

Autrement dit, comme sur les mots finis, la B-sémantique minimise la valeur sur toutes les stratégies acceptantes de $\mathcal{A}$. En particulier, peu importe les choix effectuées par Adam, la stratégie d'Evedoit garantir la présence d'une infinité d'états de Büchi.

Les S-automates de Büchi alternants (S-ABA) sont définis de la même manière, avec l'inversion entre inf et sup comme sur les mots finis. Ainsi $val_S(S) = \inf \{ val_S(\pi) : \pi \text{ branche de } S \}$, et la S-sémantique d'un S-ABA $\mathcal{A}$ est donnée par :

$$[\![\mathcal{A}]\!]_S(u) := \sup \{ val_S(S) : S \text{ est une stratégie acceptante de } \mathcal{A} \text{ sur } u \} .$$

Le but d'une stratégie dans un S-ABA est donc de maximiser la plus petite valeur enregistrée sur l'une des branches.

Une *n-exécution* d'un B-ABA est une exécution acceptante de valeur au plus n, tandis qu'une n-exécution d'un S-ABA est une exécution acceptante de valeur au moins n.

6.1.3 Automates non-déterministes

Si δ n'utilise que des disjonctions, alors Adam n'a aucun rôle dans les exécutions, et l'on dit que l'automate $\mathcal{A}$ est non-déterministe.

Les stratégies de $\mathcal{A}$ correspondent alors à ses exécutions : il suffit de choisir un élement de la disjonction à chaque pas.

L'ensemble des exécutions possibles de $\mathcal{A}$ sur un mot $u = a_0 a_1 \dots$ fixé peut être visualisé par un DAG (pour Direct Acyclic Graph) G dont les arêtes sont étiquetées par $\mathbb{C}$. Les sommets de G sont les éléments de $Q \times \mathbb{N}$, et ses arêtes sont les $(p,i) \xrightarrow{c} (q, i+1)$, où (c,q) apparaît dans $\delta(p, a_i)$. Ainsi, les exécutions de $\mathcal{A}$ sont exactement les chemins infinis dans G, qui commencent par un sommet de la forme $(p,0)$ avec $p \in In$.

Dans ce cas, on présentera souvent δ comme un ensemble, et on le notera alors $\Delta = \{(p, a, \nu, q) : (\nu, q) \text{ est une disjonction dans } \delta(p,a)\} \subseteq Q \times \mathbb{A} \times \mathbb{C} \times Q$.

On utilisera les abrévations B-NBA pour les B-automates de Büchi non-déterministes, et S-NBA pour les S-automates de Büchi non-déterministes.

Théorème 6.1.1 *Soit f une fonction de coût sur mots infinis. Les assertions suivantes sont équivalentes :*
- *f est reconnue par un B-NBA,*
- *f est reconnue par un S-NBA.*

Démonstration L'idée de cette démonstration vient de Thomas Colcombet. C'est une généralisation de la preuve de Büchi pour la complémentation des automates de Büchi classiques.

Le chapitre suivant est consacré à la preuve de ce théorème, incluant une introduction des notions algébriques nécessaires. □

En conséquence, on dira que les fonctions de coût reconnues par un B-NBA sont *régulières*.

6.1.4 Automates faibles alternants

Les automates faibles alternants classiques ont été introduits dans [MSS92], et ont prouvé leur utilité via nombreux résultats de décidabilité et de complexité en théorie des automates, sur mots infinis et arbres infinis.

On généralise ici cette notion aux fonctions de coût.

On dira qu'un B-ABA $\mathcal{A} = \langle Q, \mathbb{A}, In, F, \Gamma, \delta \rangle$ est un *B-automate faible alternant* (B-WAA) si son ensemble d'états Q peut être partitionné en des ensembles $Q_1, \dots, Q_k$, ordonnés partiellement par $\leq$, tels que :
- pour tout $q, q' \in Q_i$, $q \in F$ si et seulement si $q' \in F$;
- si $q_j \in Q_j$ peut être atteint depuis $q_i \in Q_i$ par des transitions de δ, alors $Q_j \leq Q_i$.

Cela revient à interdire l'existence d'un cycle comportant simultanément des états de Büchi (acceptants) et des états non Büchi (rejetants). Toute exécution de $\mathcal{A}$ va donc se stabiliser soit dans des états acceptants, soit dans des états rejetants.

Si de plus on peut se restreindre à des singletons pour les partitions (i.e. l'ordre porte sur les états), on dira que l'automate est *très faible*, noté B-VWAA. On peut remarquer qu'un B-ABA est un B-VWAA si et seulement si tous ses cycles sont triviaux.

Cette définition étend celle des *automates linéaires alternants* donnée dans [LT00], dans le cas classique.

6.2 Logiques de coût sur mots infinis

6.2.1 CFO et CMSO sur mots infinis

On reprend les définitions de CFO, CMSO telles qu'elles sont données dans la Section 2.4. Le fait d'interpréter maintenant ces logiques sur mots infinis n'affecte pas leur définition. La seule différence se situe dans la portée des variables : les variables de premier ordre sont à valeur dans $\mathbb{N}$ (elles étaient auparavant limitées à la longueur du mot), et les variables de second ordre sont des parties de $\mathbb{N}$, qui peuvent être finies ou infinies.

Exemple 6.2.1 *Soit $\mathbb{A} = \{a, b\}$, et $\phi = \forall^{\leq N} x.\neg a(x)$. On a pour tout mot u, $[\![\phi]\!](u) = |u|_a$, et en particulier $[\![\phi]\!](u) = \infty$ si u contient une infinité de a.*

On peut également reprendre la fonction $\varphi = \forall X.(bloc(X) \wedge a(X)) \Rightarrow (\forall^{\leq N} x.x \notin X)$ de l'Exemple 2.4.2. On aura pour tout u, $[\![\varphi]\!](u) = \mathrm{maxblock}_a(u)$, et en particulier $[\![\varphi]\!](u) = \infty$ si u contient des blocs de a de taille arbitrairement grande.

Théorème 6.2.2 *Soit f une fonction de coût sur mots infinis. Les assertions suivantes sont équivalentes :*
- *f est régulière sur mots infinis,*
- *f est reconnue par une formule de CMSO.*

Démonstration On utilisera ici la définition alternative de CMSO, qui enrichit FO avec un prédicat « $|X| \leq N$ ». On a montré dans la Section 2.4.2 que les deux formalismes sont équivalents sur mots finis, la preuve reste valide sur les mots infinis.

Des B-NBA à CMSO

Soit $\mathcal{B} = \langle Q, \mathbb{A}, In, F, \Gamma, \Delta\rangle$ un B-NBA. On veut construire une formule $\varphi_{\mathcal{B}}$ de CMSO qui reconnaît $[\![\mathcal{B}]\!]$.

Puisque les deux sémantiques (celle de CMSO et celle des B-automates) sont définies par un infimum, la formule $\varphi_{\mathcal{B}}$ doit juste exprimer qu'il existe une exécution de $\mathcal{B}$ sur u de valeur au plus N.

Soit $\Delta = \{\tau_1, \ldots, \tau_k\} \subseteq Q \times \mathbb{A} \times \mathbb{C} \times Q$ l'ensemble des transitions de $\mathcal{B}$. Pour toute transition $\tau = (p, a, \nu, q) \in \Delta$, on utilise une variable monadique X_τ pour décrire l'ensemble des positions où τ est utilisée. On note $\pi_1, \pi_{\mathbb{A}}, \pi_{\mathbb{C}}, \pi_2$ les projections de $\Delta \subseteq Q \times \mathbb{A} \times \mathbb{C} \times Q$ sur chacune de ses composantes. Pour tout $\gamma \in \Gamma$, on définit aussi $\pi_\gamma(\tau) \in \mathbb{C}_B$ la projection de $\pi_{\mathbb{C}}(\tau) \in \mathbb{C} = \mathbb{C}_B^\Gamma$ sur le compteur γ.

On définit les formules auxiliaires suivantes, qui contiennent les variables monadiques libres $X_{\tau_1}, \ldots, X_{\tau_k}$:
- **Partition** $:= \forall x. \bigvee_{1\leq i\leq k}(x \in X_{\tau_i} \wedge (\bigwedge_{j\neq i} x \notin X_{\tau_j}))$.
  Cette formule force les X_{τ_i} à partitionner l'ensemble des positions du mot.
- **Mot** $:= \forall x. \bigvee_{a\in\mathbb{A}}(a(x) \wedge \bigvee_{\pi_{\mathbb{A}}(\tau)=a} x \in X_\tau))$.
  Cette formule force les X_τ à être cohérents avec le mot d'entrée.

– **Init** $:= \bigvee_{\pi_1(\tau)\in In}(\exists x.x \in X_\tau \wedge \forall y.x \leq y)$.
Cette formule vérifie que l'exécution commence bien dans un état initial.
– **Büchi** $:= \bigvee_{\pi_2(\tau)\in F}(\forall x.\exists y.y \in X_\tau \wedge x \leq y)$.
Cette formule vérifie qu'il existe une transition vers un état de Büchi qui est prise infiniment souvent. C'est nécessaire et suffisant pour que la condition de Büchi soit vérifiée, car il existe un nombre fini de transitions.
– Pour tout $\gamma \in \Gamma$, on définit une formule
$\mathbf{Mesure}_\gamma(X) := (\forall x.x \in X \Rightarrow \bigvee_{\pi_\gamma(\tau)=\mathtt{ic}} x \in X_\tau) \wedge$
$\forall x, y, z.(x \in X \wedge y \in X \wedge x \leq z \leq y) \Rightarrow \bigvee_{\pi_\gamma(\tau)\in\{\mathtt{ic},\varepsilon\}} z \in X_\tau)$
qui vérifie que l'ensemble X marque des incréments consécutifs pour γ, en ignorant les ε intermédiaires. Notons que cette formule ne contient pas de prédicat quantitatif, c'est-à-dire que c'est une formule pure MSO, et que l'on pourra donc la nier.
– **Coût** $:= \bigwedge_{\gamma\in\Gamma}(\forall X.\mathbf{Mesure}_\gamma(X) \Rightarrow |X| \leq N)$.
Cette formule vérifie que chaque compteur effectue au plus N incréments consécutifs durant toute l'exécution.

On peut finalement définir la formule

$$\varphi_{\mathcal{B}} := \exists X_{\tau_1}, \ldots, X_{\tau_k}, \mathbf{Partition} \wedge \mathbf{Mot} \wedge \mathbf{Init} \wedge \mathbf{Büchi} \wedge \mathbf{Coût}.$$

Il est facile de vérifier que $[\![\varphi_{\mathcal{B}}]\!] = [\![\mathcal{B}]\!]$, puisque pour tout $u \in \mathbb{A}^\omega$ et $n \in \mathbb{N}$, les n-exécutions sur u sont en bijection avec les instanciations de $X_{\tau_1}, \ldots, X_{\tau_k}$ qui témoignent que $(u, n) \models \varphi_{\mathcal{B}}$.

De CMSO aux B-NBA

On veut maintenant montrer que pour toute formule φ de CMSO, on peut construire un B-NBA $\mathcal{B}_\varphi$ reconnaissant la même fonction de coût. Pour ce faire, on utilisera le Théorème 6.1.1, qui garantit l'équivalence entre les modèles B-NBA et S-NBA.

Dans la définition de CMSO, on force tous les prédicats « $|X| \leq N$ » à apparaître positivement. Si l'on nie une formule de CMSO, tous ces prédicats apparaîtront négativement. On nomme $\overline{\text{CMSO}}$ la logique correspondante, et on notera $\overline{\varphi}$ les formules de $\overline{\text{CMSO}}$.

Ceci correspond à autoriser les négations dans la syntaxe, et à définir simultanément les deux logiques duales de la manière suivante :

$$\varphi := a(x) \mid x \leq y \mid x \in X \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \exists X.\varphi \mid \neg\overline{\varphi} \mid |X| \leq N,$$
$$\overline{\varphi} := a(x) \mid x \leq y \mid x \in X \mid \overline{\varphi} \wedge \overline{\varphi} \mid \exists x.\overline{\varphi} \mid \exists X.\overline{\varphi} \mid \neg\varphi.$$

On peut ici économiser de nombreux constructeurs grâce à la présence de la négation. Par exemple les prédicat $\overline{\varphi} \vee \overline{\varphi}$ et $|X| > N$ ne sont pas nécessaires dans la syntaxe de $\overline{\text{CMSO}}$, puisqu'ils peuvent être obtenus comme $\neg(\varphi \wedge \varphi)$ et $\neg(|X| \leq N)$.

On a vu que les formules φ de CMSO correspondent naturellement aux B-automates, car la sémantique est définie comme un infimum dans ces deux formalismes.

De la même manière, on peut donner une sémantique aux formules $\overline{\varphi}$ de $\overline{\text{CMSO}}$ par

$$[\![\overline{\varphi}]\!](u) := \sup\{n \in \mathbb{N}, (u,n) \models \overline{\varphi}\}.$$

Le modèle naturel d'automates pour ces formules sera le S-automate, puisque les sémantiques sont définies par un supremum dans les deux cas.

Lemme 6.2.3 *Si φ est une formule de CMSO ou de $\overline{\text{CMSO}}$, alors $[\![\varphi]\!] \approx [\![\neg\varphi]\!]$.*

Démonstration Rappelons d'abord que si φ est une formule de CMSO, alors $\neg\varphi$ est une formule de $\overline{\text{CMSO}}$ et vice-versa.

Supposons que φ est une formule de CMSO. soit u un mot de $\mathbb{A}^\omega$. On rappelle que $[\![\varphi]\!](u) = \inf\{n \in \mathbb{N}, (u,n) \models \varphi\}$. Puisque les prédicats $|X| \leq N$ apparaissent seulement positivement φ, pour tout $k \leq n \in \mathbb{N}$, $(u,k) \models \varphi \Rightarrow (u,n) \models \varphi$. Deux cas sont alors possibles :

- Si $[\![\varphi]\!](u) > 0$, le plus grand k tel que $(u,k) \not\models \varphi$ est exactement $[\![\varphi]\!](u) - 1$. mais alors $(u,k) \not\models \varphi \iff (u,k) \models \neg\varphi$. D'où $[\![\neg\varphi]\!](u) = [\![\varphi]\!](u) - 1$.
- Si $[\![\varphi]\!](u) = 0$, alors $[\![\neg\varphi]\!](u) = \sup\emptyset = 0$.

Dans tous les cas $[\![\neg\varphi]\!](u) \leq [\![\varphi]\!](u) \leq [\![\neg\varphi]\!](u) + 1$, c'est vrai pour tout $u \in \mathbb{A}^\omega$, donc $[\![\varphi]\!] \approx [\![\neg\varphi]\!]$.

Le cas où φ est une formule de $\overline{\text{CMSO}}$ est similaire. □

On peut maintenant montrer que les assertions suivantes sont équivalentes :

1 : f est reconnue par un B-NBA.

2 : f est reconnue par un S-NBA.

3 : f est reconnue par une formule de CMSO.

4 : f est reconnue par une formule de $\overline{\text{CMSO}}$.

On a déjà $1 \Leftrightarrow 2$ par le Théorème 6.1.1, $3 \Leftrightarrow 4$ par le Lemme 6.2.3, et $1 \Rightarrow 3$ par la première partie de cette preuve.

Il nous reste à montrer que toute formule φ de CMSO peut être reconnue par un B-NBA. On montre ce résultat par induction sur φ.

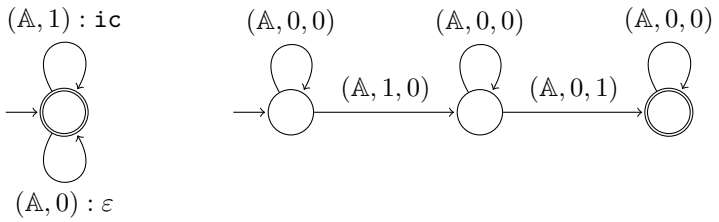
On va toujours utiliser des B-NBA pour les formules de CMSO et des S-NBA pour les formules de $\overline{\text{CMSO}}$.

Pour ce faire, on va avoir besoin de représenter des formules contenant des variables libres. Ceci est fait de manière usuelle en lisant des mots sur un alphabet étendu $\mathbb{A} \times \{0,1\}^k$, où k est le nombre de variables libres dans φ. Un mot de $\{0,1\}^\omega$ décrit la valuation pour un ensemble de positions, en marquant par des 1 les positions appartenant à l'ensemble. Les variables du premier ordre peuvent être considérées comme des singletons.

Il est maintenant facile de construire des B-NBA (en fait des automates de Büchi classiques) pour les prédicats $a(x)$, $x \leq y$, $x \in X$.

On a également besoin d'automates de Büchi pour les négations de ces prédicats, ce qui ne pose pas plus de problème. Ceci nous donne des S-NBA pour les prédicats originels.

Les B-NBA suivant reconnaissent $[\![|X| \leq N]\!]$ et $x \leq y$ (donné en exemple) :



Il nous reste à considérer les opérateurs et quantificateurs $\neg, \wedge, \exists x, \exists X$.

Par le Lemme 6.2.3, si $\varphi = \neg\psi$ est une formule de CMSO alors ψ est une formule de $\overline{\text{CMSO}}$, et $[\![\varphi]\!] \approx [\![\psi]\!]$. Il suffit donc d'utiliser l'équivalence donnée par le Théorème 6.1.1 pour convertir le S-NBA pour ψ donné par l'hypothèse d'induction, et obtenir ainsi un B-NBA pour φ. Le cas contraire est symétrique.

Les autres opérateurs correspondent à la clôture de la classe des fonctions régulière par les opérations $\min, \max,$ inf-projection et sup-projection. Plus précisément, les quantificateurs existentiels correspondent à des inf-projections dans CMSO, et à des sup-projections dans $\overline{\text{CMSO}}$.

En convertissant au besoin les B-NBA en S-NBA et vice-versa, il est facile d'utiliser la clôture de ces automates par toutes ces opérations pour obtenir le résultat. On peut lire [Col09] pour des détails sur la clôture des B-automates par $\min, \max,$ inf-projection et la clôture des S-automates par $\min, \max,$ sup-projection.

Par exemple si l'on a un automate $\mathcal{A}$ pour $\varphi(X)$ dans CMSO, on peut construire un automate $\mathcal{B}$ pour $\exists X.\varphi(X)$ en effaçant simplement la composante correspondant à X sur les transitions de $\mathcal{A}$. Cette opération correspond à une inf-projection.

En utilisant ces constructions, on peut finalement obtenir un B-NBA $\mathcal{B}_\varphi$ pour toute formule φ de CMSO.

□

6.2.2 La logique CMSO faible : WCMSO

La logique CMSO faible, notée WCMSO (pour « Weak cost MSO »), est introduite dans [VB11] sur les arbres infinis.

Comme dans le cas classique, cette logique est définie à partir de CMSO en limitant la portée des variables de second ordre aux ensembles finis. Comme on le voit dans le lemme suivant, on peut toujours remonter à une formule de CMSO en partant d'une formule de WCMSO :

Lemme 6.2.4 *Toute fonction de coût WCMSO-définissable est CMSO-définissable.*

Démonstration Soit φ une formule de WCMSO. On veut construire une formule φ' de CMSO telle que $[\![\varphi']\!] \approx [\![\varphi]\!]$. Soit $\mathbf{Fini}(X) := \exists x.\forall y.x \leq y \Rightarrow y \notin X$. T Cette formule force l'ensemble X à être fini. On peut définir φ' en ajoutant la condition $\mathbf{Fini}(X)$ pour tous les ensembles X de la formule φ. On pourra

ainsi considérer φ' comme une formule de CMSO, sans perdre pour autant la sémantique de φ, qui force les ensembles à être finis.

Plus précisément, on remplace toute occurrence de $\exists X.\psi$ par $\exists X.\mathbf{Fini}(X) \wedge \psi$, et toute occurrence de $\forall X.\psi$ par $\forall X.\mathbf{Fini}(X) \Rightarrow \psi$. Ceci nous donne une construction explicite de φ' à partir de φ. □

Les mots infinis pouvant être vus comme des arbres infinis particuliers, les définitions et résultats de [VB11] sont valides ici. En particulier, on a le théorème suivant :

Théorème 6.2.5 ([VB11]) *Les fonctions de coût reconnues par des formules de WCMSO sont exactement celles reconnues par B-WAA.*

On va voir dans la Section 8 que la réciproque est vraie sur les mots infinis.

Chapitre 7

Equivalence entre B-NBA et S-NBA

7.1 Structures algébriques sur mots infinis

L'équivalence entre B-NBA et S-NBA est une généralisation de la clôture par complémentation des automates de Büchi. Cette preuve était connue de Colcombet, mais n'était pas publiée auparavant, on l'inclut ici par souci de complétude. On va ici généraliser la preuve algébrique utilisant les ω-semigroupes et algébres de Wilke. On peut se référer à [PP95] pour une présentation de ces formalismes (cette section reprend le même contenu, de manière moins détaillée), et à [Büc62] pour la preuve originelle de complémentation de Büchi. Cette dernière n'utilise pas explicitement les structures algébriques présentées ici, mais certaines idées y sont déjà présentes. Cette construction est également proche de celle présentée dans [BC06], qui montre l'équivalence entre des modèles d'automates légèrement différents, appelés ωB-automates et ωS-automates.

7.1.1 ω-semigroupes

Les ω-semigroupe ont été définis pour étendre la notion de reconnaissance par semigroupe fini, de manière à pouvoir dire qu'un langage $L \subseteq \mathbb{A}^\omega$ est reconnu par un semigroupe fini.

Ils sont définis de la manière suivante : un ω-semigroupe est une paire $S = (S_f, S_\omega)$ muni des opérations suivantes :

- un produit binaire sur S_f, noté multiplicativement,
- un produit mixte $S_f \times S_\omega \to S_\omega$, également noté multiplicativement,
- un produit infini $\pi : S_f^{\mathbb{N}} \to S_\omega$.

L'idée derrière ces définitions est de représenter les mots finis par des éléments de S_f, et les mots infinis par des éléments de S_ω. Les divers produit correspondent donc à concaténer ces deux types de mots de différentes manières.

En conséquence, ces opérations doivent satisfaire les axiomes suivants :

– S_f muni du produi binaire est un semigroupe fini classique,
– pour tous $s, t \in S_f$ et $u \in S_\omega$, $s(tu) = (st)u$,
– pour toute suite croissante d'entiers $(k_n)_{n>0}$ et pour toute suite $(s_n)_{n\in\mathbb{N}} \in S_f^{\mathbb{N}}$, on a

$$\pi(s_0 s_1 \dots s_{k_1-1}, s_{k_1} s_{k_1+1} \dots s_{k_2} - 1, \dots) = \pi(s_0, s_1, s_2, \dots),$$

– pour tout $s \in S_f$ et $(s_n)_{n\in\mathbb{N}} \in S_f^{\mathbb{N}}$,

$$s\pi(s_0, s_1, s_2, \dots) = \pi(s, s_0, s_1, s_2, \dots).$$

Toutes ces propriétés peuvent être vues comme des règles d'associativité, et montrent que dans la suite on pourra écrire le produit $s_0 s_1 s_2 \dots$ sans ambiguité.

On supposera toujours qu'il n'y a pas d'éléments de S_ω « inutile », c'est-à-dire que tout élément de S_ω peut s'écrire comme un produit infini d'éléments de S_f.

Exemple 7.1.1 *Soit $\mathbb{A}^\infty = (\mathbb{A}^+, \mathbb{A}^\omega)$, muni des concaténations habituelles sur les mots. Alors $\mathbb{A}^\infty$ est un ω-semigroupe.*

On est confronté ici à un problème de finitude : même si le nombre d'éléments de S_f et S_ω est fini, le produit infini π ne possède pas a priori de présentation finie. En ce sens, un ω-semigroupe n'est pas a proprement parler une structure finie.

Pour remédier à ce problème, on va mettre les ω-semigroupes en relation avec une autre structure, finie cette fois-ci : les algèbres de Wilke.

7.1.2 Algèbres de Wilke

Une *algèbre de Wilke* est définie de la même manière qu'un ω-semigroupe, excepté que l'on remplace le produit infini π par une opération unaire $S_f \to S_\omega$, notée $s \mapsto s^\omega$.

Cette opération correspond à itérer une infinité de fois un élément, et doit satisfaire pour tous $s, t \in S_f$, $s(ts)^\omega = (st)^\omega$ et pour tout $n > 0$ $(s^n)^\omega = s^\omega$. On peut remarquer que cette dernière propriété implique que ω est complètement caractérisée par sa valeur sur les idempotents. En effet, étant donné $s \in S_f$, on peut toujours trouver n tel que s^n est idempotent, et on trouvera ainsi la valeur de s^ω grâce à celle de $(s^n)^\omega$.

La nouveauté est que l'on peut maintenant représenter de manière finie l'opération ω, qui remplace le produit infini en permettant de passer de S_f à S_ω. Il reste à montrer qu'elle représente de manière unique le produit infini de l'ω-semigroupe correspondant.

Cette propriété est donnée par le théorème suivant :

Théorème 7.1.2 *Pour toute algèbre de Wilke (S_f, S_ω), il existe un unique ω-semigroupe de même domaine dont le produit infini est tel que $sss\cdots = s^\omega$. En particulier, tout élément de $u \in S_\omega$ peut s'écrire st^ω avec $s, t \in S_f$.*

Ce théorème est obtenu grâce au théorème de Ramsey suivant :

Théorème 7.1.3 *Soit S un semigroupe fini et $\phi : \mathbb{A}^+ \to S$ un morphisme de semigroupes. Pour tout $u \in \mathbb{A}^\omega$, il existe une paire $(s, e) \in S^2$ avec $se = s$, $ee = e$, et une factorisation de u en $u_0u_1u_2\dots$ avec $\phi(u_0) = s$ et pour tout $n \geq 1$ $\phi(u_n) = e$.*

Réciproquement, on peut facilement associer une algèbre de Wilke à un ω-semigroupe en posant $s^\omega = \pi(sss\dots)$. On peut donc identifier les deux notions, et obtenir une présentation finie pour tout ω-semigroupe.

7.2 De B-NBA à S-NBA

Soit $\mathcal{B} = \langle Q, \mathbb{A}, F, q_0, \Gamma, \Delta\rangle$ un B-NBA sur mots infinis, et $f = [\![\mathcal{S}]\!]$. On veut construire un S-NBA $\mathcal{S}$ reconnaissant f.

7.2.1 Semigroupe d'actions

On rappelel que $\mathbb{C}_B = \{\varepsilon, \mathtt{icr}\}$ est l'ensemble des B-actions atomiques

On pose $\mathbb{C}_1 := \{\mathtt{r}, \varepsilon, \mathtt{ic}, \bot\}$: la nouvelle action $\bot$ signifie que l'on a fait trop d'incréments. Soit $\mathbb{C} = \mathbb{C}_1^\Gamma$, les éléments de $\mathbb{C}$ seront utilisés pour décrire une action globale, sur une partie d'exécution.

On donne à $\mathbb{C}_1$ la structure d'un semigroupe de stabilisation. Le produit représente la concaténation de deux actions, et l'opérateur $\sharp$ permet de répéter une action un grand nombre de fois.

$\cdot$	$\mathtt{r}$	ε	$\mathtt{ic}$	$\bot$	$\cdot^\sharp$
$\mathtt{r}$	$\mathtt{r}$	$\mathtt{r}$	$\mathtt{r}$	$\bot$	$\mathtt{r}$
ε	$\mathtt{r}$	ε	$\mathtt{ic}$	$\bot$	ε
$\mathtt{ic}$	$\mathtt{r}$	$\mathtt{ic}$	$\mathtt{ic}$	$\bot$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

On définit également un ordre par $\bot \leq \mathtt{ic} \leq \varepsilon \leq \mathtt{r}$. Attention, cet ordre n'est pas exactement celui du semigroupe de stabilisation ordonné (dans lequel $\bot \leq \mathtt{ic}$ et $\bot \leq \mathtt{r}$ suffiraient).

Le produit, l'ordre, et la stabilisation de $\mathbb{C}$ sont ensuite définies canoniquement, composante par composante.

7.2.2 Types

Soit $Sig := Q^2 \times \mathbb{C} \times \{0, 1\}$ l'ensemble des *signatures* de $\mathcal{B}$. On définit un ordre sur Sig par $(p, q, c, b) \leq (p', q', c', b')$ si $(p, q) = (p', q')$, $c \leq c'$ (pour l'ordre défini plus haut) et $b \leq b'$.

On définit l'ensemble des *types* $\mathcal{T}$ comme les parties de Sig closes vers le bas : pour tout $T \in \mathcal{T}$, $x \in T$ et $y \leq x$, on doit avoir $y \in T$.

Si X est une partie de Sig, on notera $X\downarrow$ la clôture de X pour $\leq$, i.e. $X\downarrow = \{s \in sig, \exists x \in X, s \leq x\}$.

Le but d'une signature est de décrire une portion finie d'exécution de $\mathcal{B}$. Si u est un mot fini et $n \in \mathbb{N}$, on définit la *n-signature* d'une exécution ρ de $\mathcal{B}$ sur u par (p, q, c, b), où

– p est l'état de départ de ρ, q est son état d'arrivée,
– c vaut le produit des actions de ρ si $val_B(\rho) \leq n$, et $c = \bot$ sinon,
– b vaut 1 si ρ comporte un état de Büchi, et 0 sinon.

Pour tout $n \in \mathbb{N}$, et $u \in \mathbb{A}^*$, on définit le n-type de u par

$$T_n(u) = \{n\text{-signatures des exécutions de } \mathcal{B} \text{ sur } u\} \downarrow .$$

La clôture vers le bas nous permet de considérer que si l'automate peut faire une exécution effectuant une action c, il peut également en faire une moins bonne effectuant une action $c' \leq c$ (par exemple $c' = \texttt{ic}$ et $c = \varepsilon$).

On peut remarquer que pour tout $u \in \mathbb{A}^*$ et $n \leq m$, on a $T_n(u) \subseteq T_m(u)$.

On veut étendre $\mathcal{T}$ en une algèbre de Wilke $(\mathcal{T}, \mathcal{T}_\omega)$, de manière à ce que les éléments de $\mathcal{T}_\omega$ décrivent des exécutions infinies de $\mathcal{B}$.

On pose $\mathcal{T}_\omega = 2^{Q \times \mathbb{C} \times \{0,1\}}$, que l'on peut également restreindre aux parties closes vers le bas. Intuitivement, un élément (p, c, b) de $\mathcal{T}_\omega$ décrira une exécution infinie partant de l'état p, effectuant une action globale c ($\bot$ si les valeurs des compteurs deviennent trop grandes), et voyant une infinité d'états de Büchi si $b = 1$. On peut considérer que l'action est $\bot$ s'il n'existe pas de chemin de p à q.

On définit le produit binaire sur $\mathcal{T}$ de la manière suivante :

$$t_1 \cdot t_2 := \{(p, r, c_1c_2, b_1 \vee b_2) : \exists q, (p, q, c_1, b_1) \in t_1 \text{ et } (q, r, c_2, b_2) \in t_2\} .$$

L'opérateur $\omega : \mathcal{T} \to \mathcal{T}_\omega$ est défini par

$$t^\omega = \left\{(p, c_1c^\sharp, b) : \exists q, b_1 \in Q \times \{0, 1\}\right., \left.(p, q, c_1, b_1) \in t \text{ and } (q, q, c, b) \in t\right\} .$$

Finalement, on définit le produit mixte $\cdot : \mathcal{T} \times \mathcal{T}_\omega \to \mathcal{T}_\omega$ par

$$s \cdot t = \{(p, c \cdot c', b) : (p, q, c, b_0) \in s \text{ et } (q, c', b) \in t\} .$$

Lemme 7.2.1 *$(\mathcal{T}, \mathcal{T}_\omega)$ muni de ces opérations de produit est une algèbre de Wilke.*

Soit $Ram = \left\{(s, t) \in \mathcal{T}^2 : st = s \text{ et } tt = t\right\}$, l'ensemble des paires de Ramsey sur $\mathcal{T}$. On définit l'ensemble des *paires acceptantes* Acc $\subseteq Ram$ comme étant les paires (s, t) telles qu'il existe $(q_0, c, 1) \in s \cdot t^\omega$ avec $q_0 \in In$ et pour tout $\gamma \in \Gamma$, $c(\gamma) \neq \bot$.

Cela signifie que si l'automate $\mathcal{B}$ a une exécution de n-type s suivies d'une infinité d'exécutions de n-type t sur un mot infini u, alors $[\![\mathcal{B}]\!](u) \leq 2n$: le type $s \cdot t^\omega$ contient une exécution acceptante de valeur au plus $2n$ (le 2 étant dû à la concaténation possible de deux séquences de n incréments). Ce fait sera prouvé plus en détail dans la suite.

On appelle Ref le complément de Acc dans *Ram*. On peut remarquer que Ref est clos pour l'inclusion : si $(s, t) \in$ Ref, alors pour tous $s' \subseteq s$ et $t' \subseteq t$, on a $(s', t') \in$ Ref.

7.2.3 Retour aux mots finis

La description des exécutions infinies par une algèbre de Wilke $(\mathcal{T}, \mathcal{T}_\omega)$ nous permet de reconstruire toute exécution infinie comme composition d'exécutions finies. En effet, on a vu dans la Section 7.1 que tout élément de $\mathcal{T}_\omega$ peut s'obtenir comme st^ω, avec $(s,t) \in Ram$.

Soit $t \in \mathcal{T}$, on définit la fonction de coût g_t sur $\mathbb{A}^*$ par

$$g_t(u) = \inf \{n : T_n(u) \not\subseteq t\}.$$

On montre que pour tout $t \in \mathcal{T}$, g_t est une fonction de coût régulière sur les mots finis, en construisant un B-automate $\mathcal{B}_t$ pour g_t.

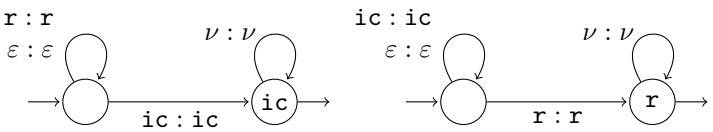
Il est suffisant de construire un B-automate pour une signature $(p,q,c,b) \notin t$ quelconque. Cet automate vérifiera que $(p,q,c,b) \in T_n(u)$. On peut ensuite procéder à l'union de tous ces automates.

On fixe donc $s = (p,q,c,b) \notin t$, et on part du B-automate $\mathcal{B}_{(p,q)} = \langle \mathbb{A}, \{p\}, \{q\}, \Gamma, \Delta \rangle$ défini à partir de $\mathcal{B}$, en changeant juste les état initiaux et finaux par p et q.

Il est simple d'étendre cet automate en $\mathcal{B}_{(p,q,b)}$, avec 2 fois plus d'états, qui se souvient si un état de Büchi a été vu, et refuse si ce n'est pas cohérent avec b.

Il reste à vérifier que l'action globale effectuée sur u est c. Pour tout $c \in \mathbb{C}_1$, on définit un automate $\mathcal{A}_c$ sur $\mathbb{C}_B^*$, qui accepte avec valeur n si l'action globale est au plus c, lorsque l'on considère une séquence de plus de n incréments comme l'action $\perp$. On peut ensuite définir les automates $\mathcal{A}_c$ avec $c \in \mathbb{C}$ en faisant les produits des automates correspondans à un seul compteur.

On présente ici les automates $\mathcal{A}_{\mathtt{ic}}$ and $\mathcal{A}_{\mathtt{r}}$ pour un compteur, avec $\nu \in \{\mathtt{ic}, \varepsilon, \mathtt{r}\}$.



On obtient finalement le B-automate voulu $\mathcal{B}_s$ en composant $\mathcal{B}_{(p,q,b)}$ avec $\mathcal{A}_c$. C'est à dire que $\mathcal{A}_c$ lira les actions effectuées par $\mathcal{B}_{(p,q,b)}$, et produira les siennes en réponse, qui seront celles de l'automate final.

On a donc montré que les g_t sont des fonctions de coûts régulières sur mots finis. Pour chaque $t \in \mathcal{T}$, on peut donc construire un S-automate $\mathcal{S}_t$ qui reconnaît g_t, en utilisant l'équivalence des B- et S-automates sur mots finis (Théorème 2.1.5).

Finalement, soit $t \in \mathcal{T}$ et $g'_t(u) := \sup \{n : T_n(u) \subseteq t\}$. On peut remarquer que $g'_t \leq g_t \leq g'_t + 1$. Cela signifie que $g_t \approx g'_t$, et donc $[\![\mathcal{A}_t]\!] \approx g'_t$.

7.2.4 Extension aux mots infinis

Soit $t \in \mathcal{T}$ et $n \in \mathbb{N}$, on pose $L_t^n := \{u \in \mathbb{A}^* : T_n(u) \subseteq t\}$.

Lemme 7.2.2 *Soit $n \in \mathbb{N}$, alors*

$$\mathbb{A}^\omega = \bigcup_{(s,t)\in Ram} L_s^n (L_t^n)^\omega. \textbf{ (1)}$$

De plus, il existe α tel que pour tout $n \in \mathbb{N}$,
- *si $(s,t) \in$ Acc et $u \in L_s^n(L_t^n)^\omega$, alors $f(u) \leq \alpha(n)$ **(2)**,*
- *si $(s,t) \in$ Ref et $u \in L_s^n(L_t^n)^\omega$, alors $f(u) > n$ **(3)**.*

Démonstration

(1) : Soit $n \in \mathbb{N}$ et $u \in \mathbb{A}^\omega$. On doit montrer qu'il existe $(s,t) \in Ram$ tel que $u \in L_s^n(L_t^n)^\omega$. On peut remarquer que $T_n : \mathbb{A}^+ \to \mathcal{T}$ est un morphisme de semigroupes. Le Théorème de Ramsey 7.1.3t nous assure l'existence d'un couple (s,t) dans $\mathcal{T}$ tel que $u = u_0u_1u_2\ldots$, $T_n(u_0) = s$ et pour tout $i \geq 1$, $T_n(u_i) = t$. On obtient donc $u \in L_s^n(L_t^n)^\omega$.

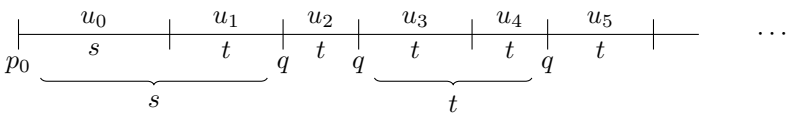
(2) : On suppose $u \in L_s^n(L_t^n)^\omega$ pour un certain $(s,t) \in$ Acc. Cela signifie qu'il existe $(q_0, c, 1) \in s \cdot t^\omega$ tel que pour tout $\gamma \in \Gamma$, $c(\gamma) \neq \bot$.

Par définition de l'algèbre de Wilke $(\mathcal{T}, \mathcal{T}_\omega)$, on a l'existence de $(p_0, q, c_0, b_0) \in s$ et $(q, q, c', 1) \in t$ avec $c_0 \cdot c'^\omega = c$.

Mais $u \in L_s^n(L_t^n)^\omega$, donc u peut être écrit $u_0u_1u_2\ldots$ avec $T_n(u_0) = s$ et pour tout $i \geq 1$, $T_n(u_i) = t$. Cela signifie qu'il existe une n-exécution de $\mathcal{B}$ de p_0 à q sur u_0 avec action c_0, et de plus pour tout $i \geq 1$ $\mathcal{B}$ peut lire u_i en faisant une boucle autour de q de valeur au plus n, d'action $c' > \texttt{ic}$, et contenant un état de BüchiĊela décrit une $2n$-exécution de $\mathcal{B}$ sur u. Le facteur 2 vient du fait que recoller deux n-exécutions peut résulter en une $2n$ exécutions. En revanche, le type des exécutions nous interdit ici de recoller trois n-exécutions avec celle du milieu de type `ic`, donc on ne dépassera jamais la valeur $2n$. On en conclut que $f(u) \leq 2n$.

(3) : On montre le résultat par contraposition. Supposons $f(u) \leq n$. Soit $u = u_0u_1u_2\ldots$ une décomposition de Ramsey de u relativement au morphisme T_n. Soit $s = T_n(u_0)$ et $t = T_n(u_i)$ pour tout $i \geq 1$, avec $tt = t$ et $st = s$.

Il existe une n-exécution acceptante ρ de $\mathcal{B}$ sur u. Soit q un état qui apparaît inifiniment souvent à la fin des u_i dans ρ.



Puisque ρ accepte avec valeur au plus n, on peut trouver $(p_0, q, c_0, b_0) \in s$ et $(q, q, c, 1) \in t$ tels que pour tout $\gamma \in \Gamma$, $c_0 \cdot c^\sharp(\gamma) \neq \bot$. Ceci implique $(s,t) \in$ Acc.

On a montré que si $f(u) \leq n$, alors toute décomposition de Ramsey (s,t) relative à T_n est dans Acc. Par contraposition, si $(s,t) \in$ Ref et $u \in L_s^n(L_t^n)^\omega$, alors $f(u) > n$.

□

Corollaire 7.2.3 *Soit g la fonction de coût sur mots infinis définie par*

$$g(u) := \sup\{n \in \mathbb{N} : \exists (s,t) \in \mathrm{Ref}, u \in L_s^n (L_t^n)^\omega\}.$$

Alors $f \approx g$.

Démonstration Soit $u \in \mathbb{A}^\omega$ tel que $f(u) > \alpha(n)$. D'après le Lemme 7.2.2, il existe $(s,t) \in \mathit{Ram}$ tel que $u \in L_s^n (L_t^n)^\omega$.

Si $(s,t) \in \mathrm{Acc}$, par le Lemme 7.2.2 on obtient $f(u) \leq \alpha(n)$ ce qui est absured, donc $(s,t) \in \mathrm{Ref}$ et $g(u) \geq n$.

Réciproquement, si $g(u) \geq n$, soit $(s,t) \in \mathrm{Ref}$ tel que $u \in L_s^n (L_t^n)^\omega$. Alors par le Lemme 7.2.2 $f(u) > n$. □

7.2.5 Construction du S-NBA $\mathcal{S}$

Soit $(s,t) \in \mathit{Ram}$, on définit $g_{s,t}(u) = \sup\{n \in \mathbb{N} : u \in L_s^n (L_t^n)^\omega\}$

Lemme 7.2.4 *Pour tous $s, t \in \mathcal{T}$, il existe un S-NBA $\mathcal{S}_{s,t}$ reconaissant $g_{s,t}$.*

Démonstration Pour tout $r \in \mathcal{T}$, on a vu que $[\![\mathcal{S}_r]\!] \approx_{\alpha_r} g'_r$ pour un certain α_r. Soit $\alpha_{s,t} = \max(\alpha_s, \alpha_t)$. L'automate $\mathcal{S}_{s,t}$ doit deviner la factorisation de u témoignant $g_{s,t}(u) \geq n$, et simuler indépendamment $\mathcal{S}_s$ sur u_0 et $\mathcal{S}_t$ sur chaque u_i, en vérifiant que chacun accepte avec valeur au plus n. Les états de Büchi sont situés à la fin de chaque u_i (et coïncident donc avec les états acceptants de $\mathcal{S}_s$ et $\mathcal{S}_t$), ce qui force l'automate à en deviner une infinité.

Si $[\![\mathcal{S}_{s,t}]\!](u) \geq \alpha_{s,t}(n)$, cela nous donne une factorisation $u = u_0 u_1 u_2 \dots$ avec $[\![\mathcal{S}_s]\!](u_0) \geq \alpha_s(n)$ et $[\![\mathcal{S}_t]\!](u_i) \geq \alpha_t(n)$ pour tout $i \geq 1$.

Cela signifie que $g'_s(u_0) \geq n$ et $g'_t(u_i) \geq n$ pour tout $i \geq 1$. D'où $g_{s,t}(u) \geq n$. On a donc $[\![\mathcal{S}_{s,t}]\!] \preccurlyeq g_{s,t}$.

Réciproquement, supposons $g_{s,t}(u) \geq n$. On a donc une factorisation $u = u_0 u_1 u_2 \dots$ avec $g_s(u_0) \geq n$ et $g_t(u_i) \geq n$ pour tout $i \geq 1$. En devinant cette factorisation, $\mathcal{S}_{s,t}$ peut effectuer une n-exécution acceptante sur u.

Finalement, $[\![\mathcal{S}_{s,t}]\!] \approx g_{s,t}$.

□

On construit enfin $\mathcal{S} = \bigcup_{(s,t)\in\mathrm{Ref}} \mathcal{S}_{s,t}$. Puisque $g = \max_{(s,t)\in\mathrm{Ref}} g_{s,t}$, et par le Corollaire 7.2.3, on peut conclure que $\mathcal{S}$ reconnaît f.

7.3 De S-NBA à B-NBA

La preuve suit le même schéma, avec quelques difficultés additionnelles dûes aux spécificités des S-compteurs.

7.3.1 Semigroupe d'actions

Ici, les effets possibles d'une exécution partielle sur un compteur sont un plus variés, et sont reflétés par $\mathbb{C}_1 := \{\Omega, \mathtt{i}, \varepsilon, \mathtt{r}, \mathtt{cr}\Omega, \mathtt{cr}, \bot\}$, où Ω signifie « beaucoup d'incréments ». On rappelle que la condition de compteurs pour les S-automates impose que chaque valeur enregistrée par un « check » est grande.

La loi de composition sur $\mathbb{C}_1$ est donc la suivante :

$\cdot$	Ω	$\mathtt{i}$	ε	$\mathtt{r}$	$\mathtt{cr}\Omega$	$\mathtt{cr}$	$\bot$	$\cdot^\omega$
Ω	Ω	Ω	Ω	$\mathtt{r}$	Ω	$\mathtt{r}$	$\bot$	Ω
$\mathtt{i}$	Ω	$\mathtt{i}$	$\mathtt{i}$	$\mathtt{r}$	$\mathtt{cr}\Omega$	$\mathtt{cr}$	$\bot$	Ω
ε	Ω	$\mathtt{i}$	ε	$\mathtt{r}$	$\mathtt{cr}\Omega$	$\mathtt{cr}$	$\bot$	ε
$\mathtt{r}$	Ω	$\mathtt{r}$	$\mathtt{r}$	$\mathtt{r}$	$\bot$	$\bot$	$\bot$	$\mathtt{r}$
$\mathtt{cr}\Omega$	$\mathtt{cr}\Omega$	$\mathtt{cr}\Omega$	$\mathtt{cr}\Omega$	$\mathtt{cr}$	$\mathtt{cr}\Omega$	$\mathtt{cr}$	$\bot$	$\mathtt{cr}\Omega$
$\mathtt{cr}$	$\mathtt{cr}\Omega$	$\mathtt{cr}$	$\mathtt{cr}$	$\mathtt{cr}$	$\bot$	$\bot$	$\bot$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Avec ordre de préférence $\Omega \geq \mathtt{i} \geq \varepsilon \geq \mathtt{r} \geq \mathtt{cr} \geq \bot$. De plus, $\varepsilon \geq \mathtt{cr}\Omega \geq \mathtt{cr}$, et les actions $\mathtt{r}$ et $\mathtt{cr}\Omega$ sont incomparables. En effet, l'ordre de préférence signifie qu'il est toujours mieux pour l'automate (il obtiendra une plus grande valeur) en remplaçant une action c par $c' \geq c$.

Or, la préférence entre $\mathtt{r}$ et $\mathtt{cr}\Omega$ peut dépendre du contexte. Précédé d'un reset, $\mathtt{rr} = \mathtt{r}$ et $\mathtt{rcr}\Omega = \bot$, donc $\mathtt{r}$ est préférable à $\mathtt{cr}\Omega$. Or, si la portion d'exécution considérée est précédée par une autre de type Ω et suivie d'une troisième de type $\mathtt{cr}$, alors $\mathtt{cr}\Omega$ devient préférable : $\Omega\mathtt{rcr} = \bot$ et $\Omega\mathtt{cr}\Omega\mathtt{cr} = \mathtt{r}$.

On ne peut donc pas comparer ces deux actions dans l'absolu.

7.3.2 Types

L'algèbre de Wilke $(\mathcal{T}, \mathcal{T}_\omega)$ des types possibles est définies de manière similaire. La seule différence étant que le n-type reflète maintenant les exécutions de valeur au moins n (au lieu du « au plus n » de la section précédente). Cela signifie qu'on autorise l'action $\mathtt{cr}$ à partir d'une valeur n des compteurs.

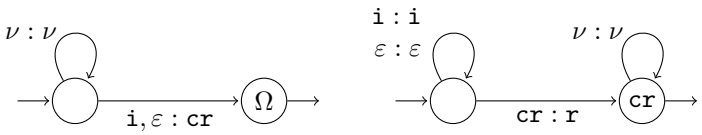
Cette fois, les paires acceptantes $\mathrm{Acc} \subseteq \mathit{Ram}$ sont les (s, t) tel qu'il existe $(q_0, c, 1) \in s \cdot t^\omega$ avec $q_0 \in \mathit{In}$ et pour tout $\gamma \in \Gamma$, $c(\gamma) \notin \{\mathtt{cr}, \mathtt{cr}\Omega, \bot\}$. Cette condition reflète le fait que l'on commence avec les compteurs à zéro, et donc l'action $\mathtt{cr}$ n'est pas autorisée en début d'exécution.

7.3.3 Automates pour les types sur mots finis

Comme précédemment, on va construire un S-automate sur mots finis pour chaque signature (p, q, c, b).

La subtilité qui s'ajoute ici est que l'on doit parfois vérifier qu'un grand nombre d'incréments ont été effectués. Au contraire, on doit parfois vérifier que l'automate effectue un $\mathtt{cr}$, sans savoir ce qui a été fait avant.

On traite donc ces cas ici en donnant les automates $\mathcal{A}_\Omega$ et $\mathcal{A}_{\mathtt{cr}}$ pour un compteur, où $\nu \in \{\mathtt{i}, \varepsilon, \mathtt{r}, \mathtt{cr}\}$.



Ces automates sont construits en accord avec les idées suivantes :

- Si c se termine par Ω, on ne peut pas utiliser les états pour vérifier que les compteurs ont atteint une grande valeur. Cependant, $\mathcal{A}_c$ peut effectuer une action `cr` à la fin du mot, comme on le voit dans $\mathcal{A}_\Omega$. De cette manière, l'exécution est acceptante si et seulement si une grande valeur est atteinte.
- si c commence par `cr`, alors le premier `cr` vu par $\mathcal{S}$ ne doit pas être recopié par $\mathcal{A}_c$, car on ne sait rien de la valeur des compteurs. On change cependant d'état, pour garder l'information qu'un `cr` a été vu.

Ainsi, si par exemple on concatène ces deux automates, et que l'action lue en entrée correspond bien à beaucoup d'incréments suivie d'un `cr`, le `cr` de sortie sera en fait effectué par le premier ($\mathcal{A}_\Omega$) et non par le deuxième ($\mathcal{A}_\mathtt{r}$), mais l'action globale sera bien $\Omega\mathtt{cr}(=\mathtt{r})$.

En combinant ces divers automates, on pourra obtenir comme précédemment un S-automate pour chaque type t. Par l'équivalence entre B- et S- automates sur mots finis, on aura finalement un B-automate $\mathcal{B}_t$ pour chaque type t.

7.3.4 Construction et correction de l'automate $\mathcal{B}$

La fin de la preuve est presque identique à celle de la section précédente. Il faut juste renverser toutes les inégalités portant sur les valeurs des fonctions, de même que les inf et sup (comme dans le passage de la sémantique B à la sémantique S).

Par exemple le Lemme 7.2.2 devient

Lemme 7.3.1 *Soit $n \in \mathbb{N}$, alors*

$$\mathbb{A}^\omega = \bigcup_{(s,t)\in Ram} L_s^n (L_t^n)^\omega. \quad \textbf{(1)}$$

De plus il existe α tel que pour tout n,

- *si $(s,t) \in$ Acc et $u \in L_s^n(L_t^n)^\omega$, alors $f(u) \geq n$ **(2)**,*
- *si $(s,t) \in$ Ref et $u \in L_s^n(L_t^n)^\omega$, alors $f(u) < \alpha(n)$ **(3)**.*

Il suffit ensuite de combiner les B-automates obtenus pour chaque type de la même manière que précédemment pour obtenir un B-NBA $\mathcal{B}$ reconnaissant $[\![\mathcal{S}]\!]$.

Chapitre 8

Des B-NBA aux B-WAA

On va montrer dans cette section que CMSO et WCMSO ont même pouvoir expressif. D'après les Théorèmes 6.2.2 et 6.2.5 et le Lemme 6.2.4, il suffit de montrer que toute fonction de coût reconnue par un B-NBA est également reconnue par un B-WAA.

Soit $\mathcal{A} = \langle Q_{\mathcal{A}}, \mathbb{A}, I_{\mathcal{A}}, F_{\mathcal{A}}, \Gamma, \Delta_{\mathcal{A}} \rangle$ un B-NBA, et $f = [\![\mathcal{B}]\!]$. On cherche à construire un B-WAA reconnaissant f.

On va faire ceci en deux étapes. On commence par transformer $\mathcal{A}$ en un autre B-NBA $\mathcal{B}$ qui reconnaît toujours f, et qui satisfait une condition structurelle (on dira que $\mathcal{B}$ est en forme normale). En effet, l'une des difficultés liées aux automates de coût est la concurrence entre la condition d'acceptation (ici Büchi), et la valeur des compteurs qui doit rester la plus basse possible. L'automate $\mathcal{B}$ aura un comportement simplifié, car la forme normale établit un lien structurel entre les actions des compteurs et la condition de Büchi.

Il sera ensuite possible d'utiliser des idées de [KV01], où la complémentation des automates de Büchi est effectuée via les automates faibles alternants.

8.1 Forme normale pour les B-NBA

On dira qu'un B-NBA $\mathcal{B} = \langle Q, \mathbb{A}, I, F, \Gamma, \Delta \rangle$ est en forme normale si pour toute transition $\tau = (p, a, \nu, q) \in \Delta$, si $p \in F$ alors $\nu = (\mathtt{r}, \mathtt{r}, \ldots, \mathtt{r})$. Autrement dit, toute transition de $\mathcal{B}$ venant d'un état de Büchi doit remettre tous les compteurs à zéro.

On va maintenant construire un B-NBA $\mathcal{B}$ en forme normale tel que $[\![\mathcal{B}]\!] \approx f$.

Pour ce faire, on part du constat suivant : pour tout $\gamma \in \Gamma$, toute exécution e de $\mathcal{A}$ de valeur finie doit vérifier l'une des deux propriétés suivantes :

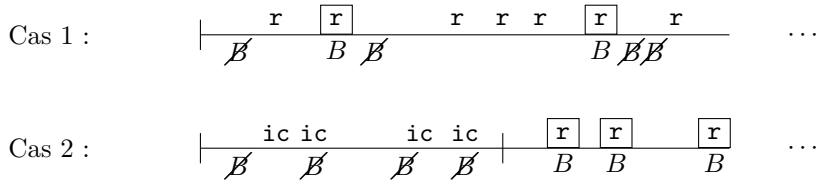
1 : il y a une infinité de resets pour γ dans e

2 : il y a un nombre fini d'incréments pour γ dans e

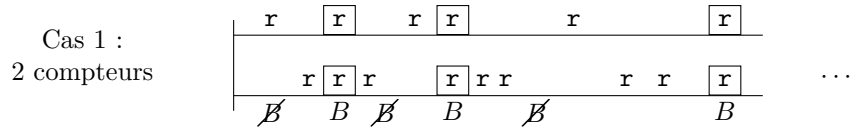
Le principe de l'automate $\mathcal{B}$ est de deviner dans quel cas chaque compteur se trouve, et d'utiliser ces propriétés pour obtenir une forme normale sans modifier la fonction de coût calculée. Dans le cas 1, on attendra d'avoir vu un reset

sur chaque compteur de type 1 pour compter le prochain état de Büchi. On ajoute une composante $1'$ pour signifier que l'on a vu un reset du compteur correspondant, et ce jusqu'au prochain état de Büchi de l'automate $\mathcal{B}$. Dans le cas 2, on pourra attendre le dernier incrément et commencer à mettre les états de Büchi après ce moment (ajouter des resets ne changera alors pas la valeur). On utilise la composante $2'$ pour signifier que l'on a deviné l'emplacement du dernier incrément, et que l'on peut maintenant prendre en compte les états de Büchi et leur adjoindre des resets.

Ces opérations de décalage des états de Büchi sont illustrées sur la figure suivante (les éléments encadrés sont ceux ajoutés dans la construction) :



Si plusieurs compteurs sont dans le cas 1, il faut attendre d'avoir vu un reset pour tous ces compteurs avant de pouvoir compter un état de Büchi :



On peut remarquer que pour chaque compteur, on insère jamais deux nouveaux resets consécutifs. De plus, si on avait une infinité d'états de Büchi au départ, il en restera une infinité après cette transformation (à condition que les compteurs considérés soient bien de type 1).

Ici, le fait que l'on travaille sur mots infinis a une importance capitale, car l'automate $\mathcal{B}$ doit deviner plusieurs propriétés sur le futur du mot : dans quel cas (1 ou 2) chaque compteur se trouve, et pour le cas 2, à partir de quel point on ne voit plus d'incrément.

Formellement, on définit $\mathcal{B} = \langle Q, \mathbb{A}, I, F, \Gamma, \Delta \rangle$, avec

- $Q = Q_{\mathcal{A}} \times \{1, 1', 2, 2'\}^{\Gamma}$
- $I = I_{\mathcal{A}} \times \{1, 2\}^{\Gamma}$
- $F = F_{\mathcal{A}} \times \{1', 2'\}^{\Gamma}$

Et finalement

$$\begin{aligned} \Delta = \quad & \{((p,x), a, g'(\nu, b), (q, x')) : (p, a, \nu, q) \in \Delta_{\mathcal{A}}, \\ & x' \in g(x, \nu, b), b = B \text{ si } (p,x) \in F \text{ et } \square \text{ sinon}\} \end{aligned}$$

où g et g' sont définis ci-dessous. Notons que $g(p, x, \nu)$ est un ensemble, donc il est possible que plusieurs (ou aucune) transitions de $\mathcal{B}$ correspondent à une transition de $\mathcal{A}$.

Le but de g est de mettre à jour la composante x en fonction de l'action ν. Il suffit de définir g pour un seul compteur γ, le cas général étant obtenu composante par composante. La composante $b \in \{B, \Box\}$) est utilisée pour décrire si l'état courant est de Büchi dans le nouvel automate : B si c'est le cas est $\Box$ sinon.

On peut maintenant définir, pour tout $b \in \{B, \Box\}$:
- $g(1, \mathtt{ic}, b) = g(1, \varepsilon, b) = \{1\}$,
- $g(1, \mathtt{r}, b) = g(1', \mathtt{r}, b) = \{1'\}$,
- $g(1', \mathtt{ic}, B) = g(1', \varepsilon, B) = \{1\}$,
- $g(1', \mathtt{ic}, \Box) = g(1', \varepsilon, \Box) = \{1'\}$
- $g(2, \nu, b) = \{2, 2'\}$ pour tout $\nu \in \{\mathtt{ic}, \varepsilon, \mathtt{r}\}$,
- $g(2', \varepsilon, b) = g(2', \mathtt{r}, b) = \{2'\}$
- $g(2', \mathtt{ic}, b) = \emptyset$.

On définit également $g'(\nu, b)$ composante par composante : c'est égal à $\mathtt{r}$ si $b = B$, et ν sinon.

Il est facile de vérifier que $\mathcal{B}$ est en forme normale : par la définition de g', les transitions partant d'un état de F effectuent un reset sur tous les compteurs.

On peut remarquer que $\mathcal{B}$ a le même ensemble de compteurs que $\mathcal{A}$, et $|Q| = |Q_{\mathcal{A}}| \times 4^{|\Gamma|}$.

Lemme 8.1.1 $[\![\mathcal{B}]\!] \approx [\![\mathcal{A}]\!]$

Démonstration Soit $u \in \mathbb{A}^\omega$, et $n \in \mathbb{N}$.

Supposons $[\![\mathcal{B}]\!](u) \leq n$. Soit ρ une n-exécution de $\mathcal{B}$ sur u. Chaque transition $((p, x), a, \nu, (q, x')$ de $\mathcal{B}$ correspond à une transition $(p, a, \nu', q) \in \Delta_{\mathcal{A}}$. Cela signifie que l'on peut définir une exécution ρ' de $\mathcal{A}$ sur u à partir de ρ.De plus, la séquence d'action ests la même, excepté que pour chaque compteur, on a pu ajouter dans ρ des resets de la manière suivante :
- Cas 1 : au plus un reset additionnel dans ρ entre deux resets de ρ'
- Cas 2 : ajout de resets après le dernier incrément.

Cela siginifie que dans tous les cas, $val_B(\rho') \leq 2 * val_B(\rho) = 2n$, d'où $[\![\mathcal{A}]\!](u) \leq 2n$. C'est vrai pour tout mot u donc $[\![\mathcal{A}]\!] \leq 2[\![\mathcal{B}]\!]$.

Réciproquement, on suppose $[\![\mathcal{A}]\!](u) \leq n$, et on choisit une n-exécution ρ de $\mathcal{A}$ sur u, partant d'un état $q_0 \in I_{\mathcal{A}}$.

On veut lui associer une n-exécution ρ' de $\mathcal{B}$ sur u.

On commence par choisir un état initial. Pour tout compteur $\gamma \in \Gamma$, soit $x_\gamma = 1$ s'il y a une infinité de $\mathtt{ic}_\gamma$ dans ρ, et 2 sinon. x_γ est appelé le *type* de γ. On peut remarquer que si $x_\gamma = 1$, alors il y a une infinité de $\mathtt{r}_\gamma$ dans ρ, sans quoi la valeur de ρ serait ∞. Soit $x = (x_{\gamma_1}, \ldots, x_{\gamma_k})$, et $q'_0 = (q_0, x)$.

Pour les compteurs γ de type 1, on n'a plus de décision à prendre, puisque $g(x, \nu, b)$ est toujours un singleton si $x \in \{1, 1'\}$, et les composantes $Q \times \{1, 1'\}$ et $Q \times \{2, 2'\}$ sont disjointes dans l'automate.

Pour les compteurs γ de type 2, il faut encore décider quand on entrera dans la composante $2'$. On peut le faire dès que le dernier incrément pour γ est passé.

Ceci définit une exécution ρ' de $\mathcal{B}$ sur u, qui ajoute des resets à ρ, ce qui a pour conséquence de diminuer la valeur maximale atteinte par les compteurs. il reste à montrer que ρ' est acceptante, pour prouver que c'est bien une n-exécution de $\mathcal{B}$ sur u.

Les états de Büchi de $\mathcal{B}$ sont donnés par $F = F_{\mathcal{A}} \times \{1', 2'\}^{\Gamma}$, ce qui signifie que toutes les composantes doivent être égales à $1'$ ou $2'$. D'après la définition de ρ', tout compteur γ de type 2 devra entrer dans la composante $2'$ à un moment, donc ces compteurs ne nous poserons pas de problème : ils peuvent seulement retarder pendant un temps fini l'apparition des états de Büchi.

On considère donc les compteurs γ de type 1. Chaque fois qu'un $\mathtt{r}_\gamma$ est observé, la composante de γ devient $1'$. On attend ensuite un état de Büchi de $\mathcal{B}$ pour revenir à 1. Puisque pour tout γ de type 1 il y a une infinité de $\mathtt{r}_\gamma$ dans ρ, on atteindra à partir de n'importe quel point une configuration où toutes ces composantes sont à $1'$. Il suffira alors d'atteindre le prochain état de Büchi de ρ pour obtenir un état de Büchi de ρ'.

ρ' comporte donc une infinité d'états de Büchi, ce qui prouve que c'est bien une n-exécution de $\mathcal{B}$ sur u, et donc $[\![\mathcal{B}]\!](u) \leq n$.

Finalement, on a obtenu $[\![\mathcal{B}]\!] \leq [\![\mathcal{A}]\!] \leq 2[\![\mathcal{B}]\!]$, donc $[\![\mathcal{B}]\!] \approx [\![\mathcal{A}]\!]$. □

8.2 Analyse des exécutions possibles de $\mathcal{B}$

Cette partie est inspirée de [KV01]. Dans cet article, Kupferman et Vardi montrent que l'on peut complémenter un automate de Büchi alternant en passant par un automate faible, avec un coût seulement quadratique en nombre d'états.

Or, on a besoin ici d'un automate sous forme normale, qu'on ne peut pas a priori obtenir à partir d'un B-ABA quelconque : il n'est plus possible pour Eve de deviner à l'avance le type des compteurs, car Adam peut avoir une influence dessus. On se restreint donc à un automate non-déterministe comme automate de départ, au lieu d'un automate alternant comme dans la preuve originale.

On va utiliser l'automate $\mathcal{B}$ en forme normale pour construire un B-WAA W reconnaissant f.

Les B-WAA étant des automates alternants, donc définis via un jeu, il est utile de voir l'acceptation de $\mathcal{B}$ comme un jeu.

Soit $u = a_0 a_1 \cdots \in \mathbb{A}^\omega$ un mot fixé. C'est Eve qui choisit toutes les transitions à prendre dans $\mathcal{B}$ pour accepter u, donc a priori il n'est pas naturel de voir l'acceptation de u par $\mathcal{B}$ comme un jeu. Cependant, on va donner un rôle à Adam, en lui demandant de prouver que Eve ne peut pas accepter le mot u. Pour ce faire, Adam devra assigner un entier (appelé le *rang*) à chaque noeud du DAG G décrivant les exécutions possibles de $\mathcal{B}$ sur u.

Soit $\mathbb{C} = \mathbb{C}_B^\Gamma$ l'ensemble des actions atomiques sur Γ.

On rappelle (voir Section 6.1.2) que le DAG $G = (V, E)$ est défini de la manière suivante : $V = Q \times \mathbb{N}$, et $E = \{((p, l), \nu, (q, l + 1)) : (p, a_l, \nu, q) \in \Delta\}$ (E est une partie de $V \times \mathbb{C} \times V$).

On définit maintenant, pour chaque ressource $n \in \mathbb{N}$ fixée, le rang de chaque sommet de G. Ces rangs vont servir à prouver que si les compteurs de l'automate $\mathcal{B}$ ne peuvent pas dépasser n, alors il n'existe pas d'exécution acceptante décrite par G.

On fixe $n \in \mathbb{N}$. Un *n-chemin* dans G est un chemin tel que la valeur de l'action obtenue en combinant les étiquettes est au plus n. Soit G' un sous-graphe de G, on dira qu'un sommet v de G' est *n-menacé* dans G' s'il n'existe pas de n-chemin infini dans G' commençant en v. On dira que v est *n-garanti* dans G' s'il n'existe pas de sommet (q, l) dans G' avec $q \in F$, atteignable depuis v par un n-chemin. Notons que dans les deux cas, il n'existe pas de n-exécution acceptante de $\mathcal{B}$ à partir de v. On définit maintenant les graphes $(G^n_k)_{k\in\mathbb{N}}$ par induction :

- $G^n_0 = G$,
- $G^n_{2i+1} = G^n_{2i} \setminus \{v : v \text{ est } n\text{-menacé dans } G^n_{2i}\}$,
- $G^n_{2i+2} = G^n_{2i+1} \setminus \left\{v : v \text{ est } n\text{-garanti dans } G^n_{2i+1}\right\}$.

Ceci nous permet de définir pour tout sommet v de G :

$$\mathbf{rang}_n(v) = \left\{k : v \in G^n_k \setminus G^n_{k+1}\right\}.$$

Intuitivement, le rang d'un sommet nous permet de nous convaincre qu'il n'existe pas de n-exécution acceptante de $\mathcal{B}$ à partir de v. Plus ce rang est élevé, et plus le nombre d'étapes pour s'en convaincre est grand. Par exemple $\mathbf{rang}_n(v) = 0$ signifie qu'il n'existe pas de n-chemin infini partant de v dans G, et donc il est facile de voir qu'il n'y a pas de n-exécution acceptante depuis v.

Lemme 8.2.1 *Pour tout $n < [\![\mathcal{B}]\!](u)$ et $i \geq 0$, il existe l_i tel que pour tout $l \geq l_i$, il y a au plus $|Q| - i$ sommets de la forme (q, l) dans G^n_{2i}.*

Démonstration On fixe $n < [\![\mathcal{B}]\!](u)$, et on prouve ce lemme par récurrence sur i. Le cas $i = 0$ est conséquence du fait que $G^n_0 = G$, et G possède au plus $|Q|$ sommets de la forme (q, l) pour tout l.

On suppose donc le résultat pour i, et on le montre pour $i + 1$.

Considérons le graphe G^n_{2i}. S'il est fini, alors G^n_{2i+1} est vide, et G^n_{2i+2} également, donc le résultat est trivialement vrai pour $i + 1$. Sinon, G^n_{2i} est infini. On montre alors qu'il existe un sommet n-garanti dans G^n_{2i+2}.

On suppose que ce n'est pas le cas, et on cherche une contradiction. Tous les sommets n-menacés de G^n_{2i} ont été retirés dans G^n_{2i+1}, donc ce dernier graphe est infini et sans feuilles. Les chemins décrits dans la suite de la preuve sont dans G^n_{2i+1}.

Soit $q_0 \in I$. Par hypothèse, $(q_0, 0)$ n'est pas n-garanti, donc il existe un n-chemin $(q_0, 0) \xrightarrow{\leq n} (q_1, l_1)$ avec $q_1 \in F$. De plus, (q_1, l_1) n'est pas une feuille, et $\mathcal{B}$ est en forme normale, donc il existe une arête $(q_1, l_1) \xrightarrow{(\mathtt{r},\mathtt{r},\ldots,\mathtt{r})} (q'_1, l_1 + 1)$. Dans ce nouvel état, tous les compteurs ont valeur 0, et par hypothèse cet état n'est pas n-garanti, donc on peut réitérer cette construction, pour obtenir un n-chemin $(q'_1, l_1 + 1) \xrightarrow{\leq n} (q_2, l_2) \xrightarrow{(\mathtt{r},\mathtt{r},\ldots,\mathtt{r})} (q'_2, l_2 + 1)$ avec $q_2 \in F$. En réitérant

cette construction à l'infini (par l'axiome du choix dépendant), on construit un chemin infini :

$$(q_0, 0) \xrightarrow{\leq n} (q_1, l_1) \xrightarrow{(\mathtt{r},\mathtt{r},\ldots,\mathtt{r})} (q'_1, l_1 + 1) \xrightarrow{\leq n} (q_2, l_2) \xrightarrow{(\mathtt{r},\mathtt{r},\ldots,\mathtt{r})} (q'_2, l_2 + 1) \xrightarrow{\leq n} \ldots$$

Ce chemin contient une infinité d'états de Büchi (les q_j pour $j > 0$), et c'est un n-chemin car il est constitué de n-chemins séparés par des resets. Il décrit donc une n-exécution de $\mathcal{B}$ sur u. Or on a $[\![\mathcal{B}]\!](u) > n$, donc une telle exécution n'existe pas.

Le raisonnement par l'absurde nous permet de conclure qu'il existe un sommet n-garanti $v = (q, l)$ dans G^n_{2i+1}.

On va montrer qu'on peut choisir $l_{i+1} = l$ pour conclure la preuve.

Puisque v est présent dans G^n_{2i+1}, il n'est pas n-menacé dans G^n_{2i}. Il existe donc un n-chemin infini π partant de v dans G^n_{2i}. Ce n-chemin est toujours présent dans G^n_{2i+1}, puisqu'aucun de ses sommets n'est n-menacé dans G^n_{2i}. De plus, le fait que v soit n-garanti dans G^n_{2i+1} implique que tous les sommets de π le sont également. Cela signifie que tous les sommets de π sont absents dans G^n_{2i+2}, et donc la largeur de G^n_{2i+2} à partir de la profondeur l est au plus $|Q| - i - 1$, par hypothèse d'induction. □

Corollaire 8.2.2 *Soit $K := 2|Q| + 1$. Pour tout $u \in \mathbb{A}^*$ et $n \in \mathbb{N}$, $[\![\mathcal{B}]\!](u) > n$ si et seulement si G^n_K est vide.*

Démonstration Par le Lemme 8.2.1, si $[\![\mathcal{B}]\!](u) > n$, alors il existe $l_{|Q|}$ tel que $G^n_{2|Q|}$ a au plus 0 sommets de la forme (q, l) pour tout $l \geq l_{|Q|}$. Cela signifie que tous les sommets de $G^n_{2|Q|}$ sont n-menacés, et donc $G_K = G_{2|Q|+1}$ est vide.

Réciproquement, si $[\![\mathcal{B}]\!](u) \leq n$, alors il existe un n-chemin π dans G comportant une infinité de sommets de F. Ceci implique que pour tout $k \in \mathbb{N}$, tous les sommets de π ne sont ni n-menacés, ni n-garantis dans G^n_k (on peut faire une récurrence simple sur k), et donc que π est présent dans tous les $(G^n_k)_{k\geq 0}$. En particulier G^n_K n'est pas vide. □

8.3 Définition du B-WAA W

On a maintenant les outils nécessaires pour décrire le B-WAA W.

Le principe de cet automate alternant est de demander à Eve de joeur les transitions de $\mathcal{B}$, tandis qu'Adam doit se fixer une ressource n, et deviner les rangs correspondants pour prouver qu'accepter avec une valeur au plus n dans $\mathcal{B}$ est impossible.

Formellement, soit $W := \langle Q_W, \mathbb{A}, Q_{in}, F_W, \Gamma, \delta \rangle$. On pose $Q_W = Q \times [0, K-1]$, $Q_{in} = I \times \{K-1\}$ et $F_W = Q \times ([0, K-1] \cap 2\mathbb{N})$. Intuitivement, si l'automate est dans l'état (q, i) en lisant la position l du mot d'entrée, cela signifie qu'Eve a amené $\mathcal{B}$ dans l'état q, et Adam a deviné que $\mathbf{rang}_n(q, l) = i$. Les états initiaux font exception, car il faut autoriser tous les rangs possibles au début, et $K - 1$ et

$K-1$ est une borne supérieure pour $\mathbf{rang}_n(q_0, 0)$, for $q_0 \in I$. On rappelle que $K = 2|Q| + 1$.

Finalement, on définit la fonction de transition $\delta : Q_W \times \mathbb{A} \to \mathcal{B}^+(\mathbb{C} \times Q_W)$ par

$$\delta(\langle q, i\rangle, a) = \begin{cases} \bigvee_{(q,a,\nu,p)\in\Delta} \bigwedge_{0\le j\le i} (\nu, \langle p, j\rangle) & \text{si } q \notin F \text{ ou } i \text{ est pair} \\ \mathsf{true} & \text{si } q \in F \text{ et } i \text{ est impair} \end{cases}$$

Si on appelle la *couleur* d'un état $col(\langle q, i\rangle) = i$, il est facile de vérifier que la couleur est décroissante le long de toute exécution de W. Les exécutions acceptantes sont donc celles qui stabilisent dans des composantes de couleur paire. Ceci montre que W est bien un B-WAA, avec une partition de Q comme union des $Q_i := Q \times \{i\}$.

8.4 Correction de W

Le rôle d'Eve est de minimiser la valeur de W, en choisissant une exécution de $\mathcal{B}$ de petite valeur sur le mot d'entrée u. Adam essaie de maximiser la valeur de W, soit en stabilisant dans des états rejetants de W (ce qui donne la valeur ∞), soit en forçant les compteurs à atteindre une valeur haute.

On fixe $u \in \mathbb{A}^\omega$, et on pose $n = [\![\mathcal{B}]\!](u) - 1$. Par le Corollaire 8.2.2, G^n_K est vide, donc $\mathbf{rang}_n$ est une fonction $V \to [0, K-1]$, décroissante le long de tous les chemins de G. Lors d'une exécution de W, Eve choisit un chemin dans G, et Adam choisit des couleurs le long de ce chemin.

On considère la stratégie σ_n d'Adam, qui consiste à étiqueter chaque sommet v de G par la couleur $\mathbf{rang}_n(v)$. On veut montrer que cette stratégie prouve que $[\![W]\!](u) \ge n + 1$.

Soit ρ une exécution compatible avec σ_n et π le chemin correspondant dans G. Premièrement, on peut s'assurer que la transition menant à true n'est jamais utilisée : si $\mathbf{rang}_n(v)$ est impair, alors v est n-garanti dans un certain G^n_{2i+1}, et v ne peut donc pas être dans F. Cela signifie que si Adam joue selon σ_n, W n'atteint jamais d'état $\langle q, i\rangle$ avec i impair et $q \in F$.

Si ρ stabilise dans une partition de couleur impaire, alors la valeur est ∞, c'est bien supérieur à $n + 1$. On suppose maintenant que ρ stabilise dans une partition de couleur paire. Puisque la couleur correspond au n-rang (Adam joue σ_n), à partir d'un certain point, tous les sommets de π sont de même rang pair $2i$, et donc sont n-menacés dans un G^n_{2i}. Or π est un chemin infini dans G^n_{2i}, donc ça ne peut pas être un n-chemin, sa valeur doit être d'au moins $n + 1$, sans quoi les sommets de π ne seraient pas n-menacés.

Dans tous les cas, la valeur de ρ est d'au moins $n + 1$, donc la stratégie σ_n prouve bien $[\![W]\!](u) \ge n + 1$. Ceci étant vrai pour tout u, on obtient $[\![W]\!] \ge [\![\mathcal{B}]\!]$.

Réciproquement, si $n = [\![\mathcal{B}]\!](u)$, on va montrer que jouer une n-exécution ρ de $\mathcal{B}$ est une stratégie de valeur au plus n pour Eve dans W. On rappelle que les compteurs de W et leurs actions sont directement copiés de ceux de $\mathcal{B}$.

La stratégie qui consiste à jouer ρ a donc pour conséquence que les compteurs de W ne dépassent jamais n. Pour Adam, Le seul moyen de mettre en échec cette stratégie est de rejeter l'entrée, c'est-à-dire de stabiliser dans une partition impaire i. Cependant, ρ contient une infinité d'états de F, donc si W stabilise dans la partiotion i, il atteindra un état $\langle q, i\rangle$ avec $q \in F$, et d'après la fonction de transition δ, cela revient à prendre une transition vers *true*, ce qui correspond à accepter (toujours avec valeur au plus n).

On a montré $[\![W]\!] \leq [\![\mathcal{B}]\!]$, on obtient donc finalement $[\![W]\!] = [\![\mathcal{B}]\!]$.

8.5 Résumé de la transformation

On peut remarquer que W et $\mathcal{B}$ reconnaissent en fait exactement la même fonction, sans nécessité de les considérer comme des fonctions de coût. De plus, le nombre d'états de W est quadratique en celui de $\mathcal{B}$, et le nombre de compteurs est préservé.

Rappelons que le passage de $\mathcal{A}$ à $\mathcal{B}$ nécessitait l'équivalence $\approx$ des fonctions de coût, et que le nombre d'états était multiplié par $4^{|\Gamma|}$.

Globalement, on a donc obtenu un B-WAA $W = \langle Q_W, \mathbb{A}, Q_{in}, F_W, \Gamma, \delta\rangle$ à partir d'un B-NBA $\mathcal{A} = \langle Q_\mathcal{A}, \mathbb{A}, I_\mathcal{A}, F_\mathcal{A}, \Gamma, \Delta_\mathcal{A}\rangle$ quelconque, avec $[\![W]\!] \approx [\![\mathcal{A}]\!]$ et $|Q_W| = \Theta(|Q|^2 2^{4|\Gamma|})$.

Par l'équivalence entre CMSO et les B-NBA (Théorème 6.2.2), et entre WCMSO et les B-WAA (Theorem 6.2.5), on obtient le résultat suivant

Théorème 8.5.1 *Toute fonction de coût régulière sur mots infinis (i.e. CMSO-définissable) est définissable par une formule de WCMSO.*

On rappelle qu'originellement, la preuve de [KV01] dans le cas classique partait d'un automate alternant $\mathcal{B}$, au lieu d'un automate non-déterministe comme il a été fait ici. On pourrait reprendre la preuve générale en partant d'un B-ABA sous forme normale (cette définition se généralise aux automates alternants) supposé fourni en entrée. Cependant, il faudrait un résultat additionnel portant sur le fait que les jeux à objectif B-Büchi sont déterminés avec mémoire finie. Il faudra de plus montrer que la mémoire M nécessaire à Eve pour gagner le jeu correspondant à l'acceptation d'un mot d'entrée u par $\mathcal{B}$ est indépendante de u. De plus, on ne pourrait pas obtenir dans ce cas une égalité $[\![W]\!] = [\![\mathcal{B}]\!]$, mais seulement une approximation $[\![W]\!] \approx_\alpha [\![\mathcal{B}]\!]$, où α dépend de la mémoire finie M définie par $\mathcal{B}$.

Chapitre 9

Fragment du Premier Ordre

9.1 CLTL sur mots infinis

Il faut ici légèrement adapter la syntaxe de CLTL, car l'opérateur Ω marquant la fin du mot n'a plus de sens pour les mots infinis.

On va donc utiliser la syntaxe suivante pour CLTL sur les mots infinis :

$$\varphi := a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \mathbf{R} \varphi \mid \varphi \mathbf{U} \varphi \mid \varphi \mathbf{U}^{\leq N} \varphi$$

On a ici omis l'opérateur $\mathbf{X}$, on pourra le définir en fonction des opérateurs présents dans la syntaxe. Pour ce faire, on utilisera cette fois-ci les variantes strictes de $\mathbf{U}$ (et $\mathbf{R}$), qui ne mettent aucune contrainte sur la position courante.

Comme sur les mots finis, on dira que $(u, n, i) \models \varphi$ si si φ est vraie sur u à partir de la position i, avec n comme valeur pour N. Tous les opérateurs conservent leur sémantique, et on définit également la sémantique de $\mathbf{R}$ (pour « Release »), dual de $\mathbf{U}$:

- $(u, n, i) \models a$ si $i \leq k$ et $a_i = a$;
- $(u, n, i) \models \varphi \wedge \psi$ si $(u, n, i) \models \varphi$ et $(u, n, i) \models \psi$;
- $(u, n, i) \models \varphi \vee \psi$ si $(u, n, i) \models \varphi$ ou $(u, n, i) \models \psi$;
- $(u, n, i) \models \varphi \mathbf{U} \psi$ s'il existe $j > i$ tel que $(u, n, i) \models \psi$ et pour tout $i < j' < j$, $(u, n, j') \models \varphi$;
- $(u, n, i) \models \varphi \mathbf{R} \psi$ si pour tout $j > i$, ou bien $(u, n, i) \models \psi$, ou bien il existe $i < j' < j$, $(u, n, j') \models \varphi$;
- $(u, n, i) \models \varphi \mathbf{U}^{\leq N} \psi$ s'il existe $j > i$ tel que $(u, n, i) \models \psi$ et pour tout $i < j' < j$ sauf au plus n positions, $(u, n, j') \models \varphi$;

On définit toujours la sémantique d'une formule φ par

$$[\![\varphi]\!](u) := \inf \{n \in \mathbb{N} : (u, n, 0) \models \varphi\}.$$

On utilisera les notations $\mathsf{true} = (\bigvee_{a \in \mathbb{A}} a)$ et $\mathsf{false} = a \wedge b$ pour a, b deux lettres distinctes de $\mathbb{A}$ (on utilise ici le fait que $\mathbb{A}$ possède au moins deux lettres).

On peut également définit les opérateurs suivants :

Futur : $\mathbf{F}\varphi = \mathsf{true}\mathbf{U}\varphi$

Toujours : $\mathbf{G}\varphi = \mathsf{false}\mathbf{R}\varphi$
Release N : $\varphi\mathbf{R}^{\leq N}\psi = \psi\mathbf{U}^{\leq N}(\varphi \vee \mathbf{G}\psi)$
Presque Toujours : $\mathbf{G}^{\leq N}\varphi = \mathsf{false}\mathbf{R}^{\leq N}\varphi$
Suivant : $\mathbf{X}\varphi = \mathsf{false}\mathbf{U}\varphi$

9.2 Pouvoir expressif de CLTL

Dans le cadre classique, Kamp a montré dans sa thèse de 1968 le théorème suivant :

Théorème 9.2.1 ([Kam68]) *Les logiques LTL et FO ont même pouvoir expressif sur mots infinis.*

On va montrer que ce théorème se généralise aux fonctions de coûts, c'est-à-dire que l'on va prouver le théorème suivant :

Théorème 9.2.2 *Soit f une fonction de coût sur mots infinis, alors f est reconnue par une formule de CFO si et seulement si f est reconnue par une formule de CLTL.*

Exemple 9.2.3 *Soit $\mathbb{A} = \{a, b, c\}$, et f la fonction de coût définie par*

$$f(u) = \begin{cases} |u|_a & \text{si } |u|_b = \infty \\ \infty & \text{sinon} \end{cases}.$$

Alors f est reconnue par la formule de CLTL $\varphi = (\mathbf{G}^{\leq N}(b \vee c)) \wedge (\mathbf{GF}b)$, et également par la formule de CFO

$$\psi = [\forall^{\leq N} x.(b(x) \vee c(x))] \wedge [\forall x.\exists y.(x < y \wedge b(x))].$$

En revanche, la fonction de coût g définie par

$$g(u) = \begin{cases} |u|_a & \text{si } |u|_b \text{ fini et pair} \\ \infty & \text{sinon} \end{cases}$$

n'est reconnaissable ni par une formule de CLTL, ni par une formule de CFO. Elle est reconnue par la formule de CMSO suivante :

$$\phi = [\forall^{\leq N} x.(b(x) \vee c(x))] \wedge [\exists X.\mathbf{pair}(X)].$$

La formule $\mathbf{pair}(X)$ vérifie que l'ensemble X repère exactement un b sur deux, en prenant le premier mais pas le dernier. Ceci garantit qu'il en existe un nombre fini et pair.

$$\begin{array}{lll} \mathbf{pair}(X) = & \wedge(\exists x \in X.\forall y.b(y) \Rightarrow x \leq y) & \text{le premier } b \text{ est dans } X \\ & \wedge[\forall x, y.(x \neq y \wedge b(x) \wedge b(y)) \Rightarrow & \\ & \quad (x \in X \wedge y \notin X) \vee (x \notin X \wedge y \in X)] & \text{un } b \text{ sur deux} \\ & \wedge(\exists x \notin X.b(x) \wedge \forall y > x, \neg b(X)) & \text{dernier } b \text{ pas dans } X \end{array}$$

9.3 De CLTL à CFO

Cette direction est la plus simple : il s'agit seulement de traduire la sémantique de tous les opérateurs de CLTL au moyen de formules du premier ordre.

On a cependant besoin de conserver des variables libres, donc on définit précisément la notion d'équivalence entre les formules de CLTL et celles de CFO comportant une variable libre.

Soit φ une formule de CLTL, et $\psi(x)$ une formule de CFO avec une variable libre x. Soit $\alpha : \mathbb{N}_\infty \to \mathbb{N}_\infty$ une fonction de correction (donc $\alpha(\infty) = \infty$). On dira que φ et $\psi(x)$ sont α-équivalentes en x si pour tout mot $u \in \mathbb{A}^\omega$ et entiers n, i,

- Si $(u, n, i) \models \psi(x)$ alors $(u, \alpha(n), i) \models \varphi$,
- Si $(u, n, i) \models \varphi$ alors $(u, \alpha(n), i) \models \psi(x)$,

On rappelle que n est la valeur pour N dans les deux formalismes, i est la valeur pour x dans le cas de $\psi(x)$, et c'est la position où la formule est évaluée dans le cas de φ. Intuitivement, on veut exprimer que si on interprète x dans $\psi(x)$ comme la position d'évaluation de la formule, alors les deux formules sont équivalentes.

Pour toute formule φ de CLTL et variable x, on veut constreuire une formule $\psi(x)$ de CFO équivalente à φ en x. Pour ce faire, on définit par induction la fonction λ_x, qui effectue cette transformation :

- $\lambda_x(a) = a(x)$,
- $\lambda_x(\varphi \wedge \psi) = \lambda_x(\varphi) \wedge \lambda_x(\psi), \quad \lambda_x(\varphi \vee \psi) = \lambda_x(\varphi) \vee \lambda_x(\psi)$,
- $\lambda_x(\varphi \mathbf{R} \psi) = \forall z > x.(\lambda_z(\psi) \vee (\exists y.x < y < z \wedge \lambda_y(\varphi))$,
- $\lambda_x(\varphi \mathbf{U} \psi) = \exists z > x.(\lambda_z(\psi) \wedge \forall y.x < y < z \Rightarrow \lambda_y(\varphi))$,
- $\lambda_x(\varphi \mathbf{U}^{\leq N} \psi) = \exists z > x.(\lambda_z(\psi) \wedge \forall^{\leq N} y.x < y < z \Rightarrow \lambda_y(\varphi))$.

Il est facile de vérifier que pour toute formule φ de CLTL, φ et $\lambda_x(\varphi)$ sont *id*-équivalentes en x. En effet, on a exactement exprimé la sémantique des opérateurs de CLTL au moyen de formules du premier ordre, donc le plus petit n satisfaisant la formule est identique dans les deux cas.

En conclusion, si φ est une formule de CLTL, on peut construire la formule de CFO suivante :« $\psi = \exists x.\lambda_x(\varphi) \wedge x = 0$ », où « $x = 0$ » est une abréviation pour « $\forall y.x \leq y$ ». ψ exprime le fait que φ est vraie en position 0, et donc $[\![\psi]\!] = [\![\varphi]\!]$.

On a donc montré le théorème suivant :

Théorème 9.3.1 *Toute fonction de coût CLTL-définissable est CFO-définissable. De plus, la traduction de CLTL vers CFO préserve les valeurs exactes des fonctions définies.*

9.4 De CFO à CLTL

On cherche ici à généraliser le Théorème de Kamp [Kam68], selon lequel FO et LTL ont même pouvoir d'expression sur mots infinis.

Au lieu de chercher à traduire directement une formule de CFO dans CLTL, on va commencer par étendre CLTL par des opérateurs portant sur le passé. En

effet, tous les opérateurs de CLTL portent sur le futur du mot, et l'on a besoin de plus de symétrie pour pouvoir refléter le comportement de CFO.

On définit donc les analogues « passé » de $\mathbf{U}, \mathbf{R}, \mathbf{X}, \mathbf{F}, \mathbf{G}$: respectivement $\mathbf{S}, \mathbf{Q}, \mathbf{Y}, \mathbf{P}, \mathbf{H}$, ainsi que les extensions quantitatives $\mathbf{S}^{\leq N}$, $\mathbf{Q}^{\leq N}$, $\mathbf{H}^{\leq N}$.

Soit $\mathrm{CLTL_P}$ la logique CLTL étendue par ces opérateurs passé.

On dit qu'une formule de $\mathrm{CLTL_P}$ est *pure passé* (resp. *pure futur*) si elle utilise uniquement des opérateurs passés (resp. futur) et tous les atomes sont sous au moins un opérateur temporel. Une formule est *pure présent* si elle ne contient aucun opérateur temporel.

Cela signifie que la valeur d'une formule pure passé (resp. présent, futur) dépend seulement des positions situées avant (resp. à, après) la position à partir de laquelle le mot est évalué. On dira qu'une formule est *pure* si elle est pure passé, pure présent ou pure futur.

Il s'avère que l'on peut toujours séparer une formule de $\mathrm{CLTL_P}$en formules pures :

Théorème 9.4.1 (Séparation) *Toute formule de $CLTL_P$ est équivalente à une combinaison booléenne de formules pures.*

La preuve de ce théorème est technique est requière une analyse d'un certain nombre de cas, décrivant les différentes manières dont les opérateurs passés peuvent se trouver sous un opérateur futur (et vice-versa). La preuve détaille est donnée en annexe, on donne ici les idées principales.

La structure de la preuve est empruntée à [GHR94], mais il faut prendre en compte de nouveaux phénomènes liées aux opérateurs quantitatifs. En particulier l'impossibilité d'utiliser la négation introduit de nouvelles difficultés. Pour parvenir à séparer chaque formule, il faut à de nombreuses reprises utiliser le fait qu'on ne cherche pas à transcrire exactement le nombre d'erreurs, mais seulement conserver le caractère borné ou non. La relation d'équivalence des fonctions de coûts est donc cruciale ici, car cette séparation est impossible si on cherche à conserver exactement le nombre d'erreurs.

On donne un exemple pour illustrer cette idée :

Exemple 9.4.2 *Soit $\mathbb{A} := \{a, b, c, d\}$. On considère la formule $\varphi = (b\mathbf{U}^{\leq N}c)\mathbf{S}a$ de $CLTL_P$. Alors φ est équivalente à*

$$[(b\mathbf{S}^{\leq N}(c \vee a))\mathbf{S}a] \wedge [(b\mathbf{U}^{\leq N}c) \vee c \vee (\mathbf{Y}a)]$$

Cette formule factorise le mot d'entrée en blocs séparés par des c, à partir du dernier a dans le passé. La première clause vérifie que chaque bloc contient au plus N lettres qui ne sont pas des b (donc des lettres d). La seconde clause vérifie que la position précédant la postion courante vérifie soit $b\mathbf{U}^{\leq N}c$, soit a.

La preuve procède ensuite par une récurrence sur la *profondeur d'alternation* : le nombre maximal d'alternations entre opérateurs passés et futurs imbriqués, ainsi que sur le rang de quantification de la formule.

Chaque pas d'induction introduit une nouvelle fonction de coût (le nombre d'erreurs peut être mis au carré), mais le fait que la profondeur d'alternation et le rang de quantification de la formule soit borné nous garantit qu'on obtient bien une formule α-équivalente, pour un certain α.

On peut maintenant décrire une traduction de CFO vers CLTL. Cette partie est adaptée de [HR05]. Elle procède par récurrence sur le rang de quantification de la formule originale, et utilise le Théorème de Séparation. Dans cette partie, le côté quantitatif n'ajoute pas de difficulté particulière, les nouveaux comportements ayant été traités dans le Théorème de Séparation.

Proposition 9.4.3 *Toute formule de CFO peut être effectivement traduite en une formule de CLTL.*

Démonstration

Comme dans la Section 9.3, on va en fait utiliser des formules de CFO de la forme $\varphi(x)$, comportant une variable libre x qui décrit la position courante dans le mot. On va aussi supposer que les formules de CFO peuvent utiliser un ensemble arbitraire de prédicats unaires $P_1, \ldots, P_n$ (par exemple les prédicats de lettres) et de leurs négations. Le but est de transformer une telle formule en une formule de $\mathrm{CLTL_P}$, qui évalue le mot à partir de la position x, et qui peut utiliser les mêmes prédicats unaires $P_1, \ldots, P_n, \neg P_1, \ldots, \neg P_n$. Les opérateurs passés seront nécessaires ici, car x peut ne pas se trouver au début du mot. On pourra ainsi traduire toute formule close φ de CFO en une formule de $\mathrm{CLTL_P}$, en appliquant simplement la transformation précédente à la formule $\varphi(x) \wedge x = 0$.

Soit $\mathrm{qr}(\varphi(x))$ le rang de quantification de $\varphi(x)$, c'est-à-dire le nombre de quantificateurs imbriqués dans $\varphi(x)$.

La preuve procède par récurrence sur $\mathrm{qr}(\varphi(x))$.

Si $\mathrm{qr}(\varphi(x)) = 0$, alors $\varphi(x)$ est une combinaison booléenne d'atomes de la forme $x \leq x$ et $P_i(x)$, donc clairement équivalente à une formule de $\mathrm{CLTL_P}$.

On suppose le résultat vrai pour des rangs de quantification au plus k. Il suffit donc de montrer le résultat pour des formules de la forme $\exists y.\varphi(x,y)$, $\forall y.\varphi(x,y)$, $\forall^{\leq N} y.\varphi(x,y)$, puisqu'une formule de rang $k+1$ est une combinaison booléenne de telles formules, et de formules de rang inférieur.

Le principe général est de retirer la variable x de $\varphi(x,y)$ afin de pouvoir utiliser l'hopthèse d'induction sur la formule résultante, qui n'a plus que y comme variable libre..

Pour ce faire, on commence par définir pour chaque $S \subseteq [1,n]$ une formule $\varphi^S(x,y)$, qui remplace dans $\varphi(x,y)$ chaque $P_i(x)$ par $\top$ si $i \in S$ et par $\bot$ si $i \notin S$.

Ainsi $\varphi(x,y)$ est équivalente à

$$\bigwedge_{S \subseteq [1,n]} ((\bigwedge_{i \in S} P_i(x) \wedge \bigwedge_{i \notin S} \neg P_i(x)) \Rightarrow \varphi^S(x,y)).$$

Puisque les $P_i(x)$ et leurs négations sont disponibles dans les formules de $\mathrm{CLTL_P}$, et que les quantifications sur y commutent avec les conjonctions et

les implications de cette formule, il suffit de montrer le résultat pour chaque $\varphi^S(x,y)$. Or ces formules ne contiennent pas de prédicats de la forme $P_i(x)$, portant sur la variable x. Soit $\varphi'(x,y)$ une telle formule.

On remplace dans $\varphi'(x,y)$ les occurences de $z < x$ by $\mathrm{Pos}_<(z)$, $z = x$ par $\mathrm{Pos}_=(z)$ et $z > x$ par $\varphi'(x,y)$, pour toute variable z. Autrement dit, on abstrait la comparaison avec x au moyen de trois nouveaux prédicats. Cela nous donne une formule $\varphi''(y, \mathrm{Pos}_<, \mathrm{Pos}_=, \mathrm{Pos}_>)$, utilisant ces nouveaux prédicats, mais où x n'est plus du tout utilisé.

Dans la suite, on se restreindra aux structures où les prédicats ont bien la signification qu'on leur a donné : $\mathrm{Pos}_<(z)$ est vrai jusqu'à un certain point $x-1$, $\mathrm{Pos}_=(z)$ est vrai seulement en x, et $\mathrm{Pos}_>(z)$ est vrai ensuite. Il est facile d'imposer cette condition par une formule, et donc de retrouver la valeur de x à partir de la valeur de ces prédicats.

Par hypothèse d'induction, on peut obtenir une formule ψ de $\mathrm{CLTL_P}$, équivalente à φ'' en y (sur les structures pertinentes), et utilisant les prédicats $\mathrm{Pos}_<, \mathrm{Pos}_=, \mathrm{Pos}_>$.

On peut maintenant remarquer que

- $\exists y.\varphi'(x,y)$ est équivalent à $\psi' = \mathbf{P}\psi \vee \psi \vee \mathbf{F}\psi$;
- $\forall y.\varphi'(x,y)$ est équivalent à $\psi' = \mathbf{H}\psi \wedge \psi \wedge \mathbf{G}\psi$;
- $\forall^{\leq N} y.\varphi'(x,y)$ est équivalent à $\psi' = \mathbf{H}^{\leq N}\psi \wedge \mathbf{G}^{\leq N}\psi$, avec une fonction de correction $\alpha(m) = 2m+1$.

Dans tous les cas ψ' est évaluée en position x, et utilise encore les prédicats $\mathrm{Pos}_<, \mathrm{Pos}_=, \mathrm{Pos}_>$ qui comparent la position courante avec x.

On utilise maintenant le Théorème de Séparation : ψ' peut être transformée en combinaison booléenne de formules pures. On peut maintenant remplacer $(\mathrm{Pos}_<, \mathrm{Pos}_=, \mathrm{Pos}_>)$ par $(\top, \bot, \bot)$ (resp. $(\bot, \top, \bot), (\bot, \bot, \top)$) dans les formules pur passé (resp. pur présent, pur futur). Ceci nous permet d'obtenir une formule ψ'' de $\mathrm{CLTL_P}$ utilisant seulement les prédicats originels $P_1, \ldots, P_n$, et équivalente à la formule de CFO originelle en x. Ceci conclut la récurrence.

Cependant, on voulait originellement obtenir une formule de CLTL sans opérateurs relatifs au passé. Or, on peut obtenir ici une formule séparée, évaluée à la première position du mot, en traduisant la formule $\varphi(x) \wedge x = 0$. Les formules pur passé peuvent être trivialement transformées en true ou false dans cette formule séparée, car elles portent alors sur le mot vide. On obtient donc la formule de CLTL voulue.

□

9.5 Démonstration du Théorème de Séparation 9.4.1

On va montrer comment éliminer l'imbircation d'opérateurs futurs sous des opérateurs passés. Par symétrie, il n'est pas nécessaire de traiter le cas réciproque.

On va utiliser pour la concision les notations $\overline{\mathbf{S}}, \overline{\mathbf{S}^{\leq N}}, \overline{\mathbf{U}}, \overline{\mathbf{U}^{\leq N}}, \overline{\mathbf{R}}$ pour les versions larges des opérateurs correspondants. Ceci correspond à les décaler d'une

position, la position courante étant maintenant prise en compte. Pour être précis, on définit $A\overline{\mathbf{U}}B = B \vee (A \wedge \mathbf{X}(A\mathbf{U}B))$. on obtient donc $\mathbf{X}(A\overline{\mathbf{U}}B) \equiv A\mathbf{U}B$.

On dira qu'une formule est *séparée* si c'est une combinaison booléenne de formules pures.

9.5.1 Opérations locales

On va commencer par enlever les imbrications à un niveau donné, en traitant les sous-formules comme des boîtes noires. A cause de la positivité demandée, il est important de conserver la positivité de toutes les sous-formules, dans la mesure où elles peuvent contenir des opérateurs quantitatifs.

Le lemme suivant nous est donné dans la preuve classique de [GHR94], il ne pose pas plus de diffculté ici.

Lemme 9.5.1 ([GHR94])
- $c\mathbf{U}(a \vee b) \Leftrightarrow (c\mathbf{U}a) \vee (c\mathbf{U}b)$ *;*
- $c\mathbf{S}(a \vee b) \Leftrightarrow (c\mathbf{S}a) \vee (c\mathbf{S}b)$ *;*
- $(a \wedge b)\mathbf{U}c \Leftrightarrow (a\mathbf{U}c) \wedge (b\mathbf{U}c)$ *;*
- $(a \wedge b)\mathbf{S}c \Leftrightarrow (a\mathbf{S}c) \wedge (b\mathbf{S}c)$ *;*

De plus, ce lemme est toujours vrai avec les variantes quantitatives $\mathbf{U}^{\leq N}$ and $\mathbf{S}^{\leq N}$. Les deux premiers cas (sur la disjonction) conservent la valeur exacte du nombre d'erreurs, tandis que les deux derniers (sur la conjonction) entrainent une fonction de correction $\alpha(n) = 2n$. Ce lemme est également vrai pour les variantes Release $\mathbf{R}$ and $\mathbf{Q}$.

Les cas classiques d'imbrication déjà traités dans [GHR94] sont :

1. $q\mathbf{S}(a \wedge (A\mathbf{U}B))$
2. $q\mathbf{S}(a \wedge (B\mathbf{R}A))$
3. $(q \vee (A\mathbf{U}B))\mathbf{S}a$
4. $(q \vee (B\mathbf{R}A))\mathbf{S}a$
5. $(q \vee (A\mathbf{U}B))\mathbf{S}(a \wedge (A\mathbf{U}B))$
6. $(q \vee (B\mathbf{R}A))\mathbf{S}(a \wedge (B\mathbf{R}A))$

C'est-à-dire qu'une formule qui retire l'imbrication des opérateurs futurs sous un opérateur passé dans ces formules est donnée explicitement pour chacun de ces cas.

Pour pouvoir réutiliser la preuve de [GHR94], il faut néanmoins examiner attentivement l'emploi des négations. Malgré le fait qu'elles soient utilisées, on peut remarquer que la positivité de chaque sous-formule est conservée : on applique toujours en réalité des doubles négations. Ceci nous permet de reprendre telles quelles les preuves de ces cas.

On peut également constater que dans [GHR94], deux cas supplémentaires sont traités : $(q \vee (A\mathbf{U}B))\mathbf{S}(a \wedge (B\mathbf{R}A))$ et $(q \vee (B\mathbf{R}A))\mathbf{S}(a \wedge (A\mathbf{U}B))$. En réalité, on n'aura pas besoin de considérer ces cas ici, car on considére que $A\mathbf{U}B$ n'est

pas une sous formule de $B\mathbf{R}A$, tandis que dans [GHR94], $B\mathbf{R}A$ était remplacé par $\neg(A\mathbf{U}B)$ (ce que l'on ne peut pas faire ici).

On remarque donc que faire attention à la positivité des sous-formules nous permet de réduire le nombre de cas à étudier, ce qui peut déjà constituer une légère amélioration de la preuve de [GHR94] dans le cas classique.

Il nous faut maintenant examiner les variantes quantitatives, et traiter les cas suivants :

1. (i) $q\mathbf{S}(a \wedge (A\mathbf{U}^{\leq N}B))$
   (ii) $q\mathbf{S}^{\leq N}(a \wedge (A\mathbf{U}B))$
   (iii) $q\mathbf{S}^{\leq N}(a \wedge (A\mathbf{U}^{\leq N}B))$
2. (i) $q\mathbf{S}^{\leq N}(a \wedge (B\mathbf{R}A))$
3. (i) $(q \vee (A\mathbf{U}^{\leq N}B))\mathbf{S}a$
   (ii) $(q \vee (A\mathbf{U}B))\mathbf{S}^{\leq N}a$
   (iii) $(q \vee (A\mathbf{U}^{\leq N}B))\mathbf{S}^{\leq N}a$
4. (i) $q \vee (B\mathbf{R}A))\mathbf{S}^{\leq N}a$
5. (i) $(q \vee (A\mathbf{U}^{\leq N}B))\mathbf{S}(a \wedge (A\mathbf{U}^{\leq N}B))$
   (ii) $(q \vee (A\mathbf{U}B))\mathbf{S}^{\leq N}(a \wedge (A\mathbf{U}B))$
   (iii) $(q \vee (A\mathbf{U}^{\leq N}B))\mathbf{S}^{\leq N}(a \wedge (A\mathbf{U}B))$
6. (i) $(q \vee (B\mathbf{R}A))\mathbf{S}^{\leq N}(a \wedge (B\mathbf{R}A))$

On cherche donc dans chaque cas à expliciter une formule équivalente, dans laquelle les opérateurs explicites passés et futurs n'apparaissent plus imbriqués.

Les cas 1.i à 2.i sont assez simples : le comportement est le même que dans les cas classiques, excepté le fait que certains intervalles du mots sont coupées en deux (passé et futur). On peut donc doubler le nombre d'erreurs dans le pire cas. Voici le détail de ces cas :

1.i : $q\mathbf{S}(a \wedge (A\mathbf{U}^{\leq N}B))$

Soit t la position courante et y la position atteinte par le Until.
La formule 1.i est équivalente à

$$\begin{array}{ll} & (q\mathbf{S}a) \wedge (A\mathbf{S}^{\leq N}a) \wedge (A\mathbf{U}^{\leq N}B) \quad : \ t < y \\ \bigvee & (q\mathbf{S}a) \wedge (A\mathbf{S}^{\leq N}a) \wedge B \quad : \ t = y \\ \bigvee & q\mathbf{S}(B \wedge q \wedge (A\mathbf{S}^{\leq N}a) \wedge (q\mathbf{S}a)) \quad : \ y < t \end{array}$$

Remarquons que l'on peut contracter le B courant et la formule future $A\mathbf{U}^{\leq N}B$ en une unique formule $A\overline{\mathbf{U}^{\leq N}}B := B \vee (A\mathbf{U}^{\leq N}B)$. On effectuera ce genre de simplifications dans la suite, en gardant à l'esprit qu'il est toujours possible de revenir à une formule séparée (combinaison booléenne de pur passé, pur présent et pur futur).

La fonction de correction est ici $\alpha(n) = 2n + 1$. Le $+1$ est dû au premier cas, où l'on ne demande pas A en position t, ce qui peut causer une erreur supplémentaire.

1.ii : $q\mathbf{S}^{\leq N}(a \wedge (A\mathbf{U}B))$

Equivalente à

$$\bigvee \begin{array}{l} (q\mathbf{S}^{\leq N}a) \wedge (A\mathbf{S}a) \wedge (A\overline{\mathbf{U}}B) \\ q\mathbf{S}^{\leq N}(B \wedge (A\mathbf{S}a) \wedge (q\mathbf{S}^{\leq N}a)) \end{array}$$

Correction $\alpha(n) = 2n + 1$.

1.iii : $q\mathbf{S}^{\leq N}(a \wedge (A\mathbf{U}^{\leq N}B))$

Equivalente à

$$\bigvee \begin{array}{l} (q\mathbf{S}^{\leq N}a) \wedge (A\mathbf{S}^{\leq N}a) \wedge (A\overline{\mathbf{U}^{\leq N}}B) \\ q\mathbf{S}^{\leq N}(B \wedge (A\mathbf{S}^{\leq N}a) \wedge (q\mathbf{S}^{\leq N}a)) \end{array}$$

Correction $\alpha(n) = 2n + 1$.

2.i : $q\mathbf{S}^{\leq N}(a \wedge (B\mathbf{R}A))$

Equivalente à

$$\bigvee \begin{array}{l} (q\mathbf{S}^{\leq N}a) \wedge (A\mathbf{S}a) \wedge (B\overline{\mathbf{R}}A) \\ q\mathbf{S}^{\leq N}(B \wedge (A\mathbf{S}a) \wedge (q\mathbf{S}^{\leq N}a)) \end{array}$$

Correction $\alpha(n) = 2n + 1$.

3.i : $(q \vee (A\mathbf{U}^{\leq N}B))\mathbf{S}a$

Cette formule exprime qu'il y a une position y dans le passé où a est vrai, et à partir de cette position, le mot s'écrit comme une concaténation de blocs de la forme : $qqqqqqqAAAAAxAAxAAB$, et ce jusqu'à la position courante (le dernier bloc pouvant dépasser dans le futur). Le nombre d'erreurs x doit être bornés par N dans chaque segment de A.

Soit $\varphi = [A\mathbf{S}^{\leq N}(q\overline{\mathbf{S}}(B \vee a)]\mathbf{S}a$. Alors φ est une formule pur passé, qui vérifie que chaque bloc est correct.

pour ce qui est du présent et du futur, deux cas peuvent se produire : ou bien $A\mathbf{U}^{\leq N}B$ est vrai à la position $t-1$, ou bien le dernier bloc ne contient que des q. La formule 3.i est donc équivalente à $\varphi \wedge [(A\overline{\mathbf{U}^{\leq N}}B)) \vee (q\overline{\mathbf{S}}(B \vee a))]$.

Le dernier bloc pouvant être coupé en deux, on obtient une fonction de correction $\alpha(n) = 2n$.

3.ii : $(q \vee (A\mathbf{U}B))\mathbf{S}^{\leq N}a$

On doit maintenant compter le nombre global d'erreurs, et non dans chaque bloc indépendamment. On distingue deux sortes de blocs : les « bons blocs » de la forme $BqqqqqqqqqAAAAAAAAAAB$, et les « mauvais blocs », qui sont autorisés à faire des erreurs durant les segments de q (la fin d'un tel segment étant définie comme la première position où $A\mathbf{U}B$ est vraie).

On veut sommer le nombre de q manquants sur tout le mot. On va borner ce nombre en bornant le nombre maximal d'erreurs dans un mauvais bloc, ainsi que le nombre total de mauvais blocs. Le nombre d'erreurs dans un mauvais bloc est défini comme le nombre de q manquants avant la première position où $A\mathbf{U}B$ is true.

On peut remarquer qu'un bon bloc doit commencer par B ou a, suivi d'un segment de q, puis d'un segment of A, jusqu'au prochain B ou à la position courante.

Soit $\varphi_1 = [A\mathbf{S}(q\overline{\mathbf{S}^{\leq N}}(B \vee a))]\mathbf{S}a$, cette formule compte le nombre d'erreurs dans un mavais bloc

On a aussi besoin de borner le nombre total de mauvais blocs. Pour ce faire, on aimerait écrire une formule comme $[B \Rightarrow A\mathbf{S}(q\overline{\mathbf{S}}(B \vee a))]\mathbf{S}^{\leq N}a$. Cependant, il faut faire attention car la prémisse B de l'implication est en réalité sous une négation dans cette formule. Puisque B peut en général contenir des opérateurs quantitatifs, ceci n'est pas permis.

A la place, on va vérifier des conditions locales dans chaque bloc. Le nombre total d'erreurs locales sera ensuite compris entre le nombre de mauvais blocs et deux fois le nombre d'erreurs totales de la formule originelle.

Soit $\varphi_2 = ([(q \vee B) \wedge \mathbf{Y}(q \vee B)] \vee [(A \vee B) \wedge (\mathbf{Y}(A \vee q \vee B))])\,\mathbf{S}^{\leq N}a$. Cette formule décrit la condition locale qui est vérifiée dans tout bon bloc, et compte le nombre de fois où cette condition n'est pas vérifiée.

Chaque erreur dans un segment de q est responsable d'au plus deux erreurs dans φ_2. De plus, chaque mauvais bloc commet au moins une erreur dans φ_2. L'encadrement décrit plus haut est donc vérifié.

Remarquons que pour la concision, on ne mentionn epas a dans les conditions locales. Ceci peut ajouter juste une erreur en tout, immédiatement après le a qui marque le début du premier bloc.

La formule 3.ii est finalement équivalente à $\varphi' = \varphi_1 \wedge \varphi_2 \wedge [(A\overline{\mathbf{U}}B) \vee (q\mathbf{S}^{\leq N}(B \vee a))]$. Remarquons qu'on autorise le bloc contenant la position courante à être un mauvais bloc, car on demande seulement $(q\mathbf{S}^{\leq N}(B \vee a))$ à la position courante (dans la cas où $A\overline{\mathbf{U}}B$ n'est pas satisfait). Ceci introduit au maximum N erreurs supplémentaires.

On obtient donc une fonction de correction $\alpha(n) = 2n^2 + n + 1$.

Exemple 9.5.2 *On illustre ce cas fondamental par un exemple. Pour conserver les notations, on travaillera sur l'alphabet $\mathbb{A} = \{a, q, A, B, e\}$, où e est une nouvelle lettre, nécessaire pour commettre des erreurs. Soit*

$$u = BeaAaqqAAABeeeqqAqAABAAeAA|A|ABAq.$$

Les lignes verticales dans u marquent la position t où l'on veut évaluer la formule. Par exemple $[\![A \wedge \mathbf{X}A]\!](u) = 0$ car il y a effectivement un A en position t, et un A en position $t + 1$.

On veut maintenant évaluer la formule 3.ii sur u, autrement dit trouver la valeur de $[\![q \vee (A\mathbf{U}B))\mathbf{S}^{\leq N}a]\!](u)$. Le dernier a avant t est en position 4 dans u (la première lettre est en position 0).

La formule compte le nombre total d'erreurs, marquées ici en gras :

$$BeaAaqqAAAB\mathbf{eee}qq\mathbf{A}qAAB\mathbf{AAe}AA|A|ABAq,$$

d'où $[\![q \vee (A\mathbf{U}B))\mathbf{S}^{\leq N}a]\!](u) = 7$.

On évalue maintenant la formule séparée décrite plus haut. Premièrement, $[\![\varphi_1]\!](u) = 4$ *: c'est le nombre maximal d'erreurs dans un même bloc. On marque maintenant en gras les positions où les contraintes locales décrites par* φ_2 *ne sont pas respectées :* $BeaAa\mathbf{q}qAAAB\mathbf{eeeq}qA\mathbf{q}AABAA\mathbf{e}\mathbf{A}A|A|ABAq$*. Ceci nous donne* $[\![\varphi_2]\!](u) = 8$*. Finalement, la dernière clause de la formule est vérifiée avec* $N = 0$*, puisque* $A\overline{\mathbf{U}}B$ *est vraie en position* t*.*

Puisque la conjonction correspond à un maximum, on obtient $[\![\varphi']\!](u) = 8$*. On a bien sûr* $7 \leq \alpha(8)$ *et* $8 \leq \alpha(7)$*.*

3.iii : $(q \vee (A\mathbf{U}^{\leq N}B))\mathbf{S}^{\leq N}a$

Un bon bloc est maintenant autorisé à comporter N erreurs dans les segments de A. On veut borner le nombre total d'erreurs dans les segments de q segments, mais les erreurs dans les segments de A doivent seulement être borné dans chaque bloc, et non globalement.

On définit donc $\varphi_1 = [A\mathbf{S}^{\leq N}(q\overline{\mathbf{S}^{\leq N}}(B \vee a))]\mathbf{S}a$, qui borne localement le nombre d'erreurs dans chaque mauvais bloc.

La difficulté ici est d'exprimer des contraintes locales sur les bons blocs, car ils sont quand même autorisés à commettre des erreurs surles segments de A.

L'astuce est de considérer que les segments de q se terminent toujours par un q et non par une erreur (ou un B s'il n'y a pas de q). On peut compter les erreurs suivantes comme étant dans le segment de A, quitte à doubler la borne finale. Il est en effet plus intéressant de compter le maximum d'erreurs comme étant locales au bloc plutôt que globales.

Les contraintes locales vont donc être de la forme « ou bien on est dans un segment de A segment (incluant des erreurs), ou bien q est vraie dans la position courante et celle qui précède »

Soit $\varphi_2 = [(A\overline{\mathbf{S}^{\leq N}}(q \vee B)) \vee ((q \vee B) \wedge \mathbf{Y}(q \vee B))]\mathbf{S}^{\leq N}a$.

Comme précédemment, le nombre d'erreurs comptées par l'opérateur principal $\mathbf{S}^{\leq N}$ de φ_2 va être compris entre le nombre de mauvais blocs et le double du nombre total d'erreurs de la formule originelle.

Finalement, la formula 3.iii est equivalente à $\varphi_1 \wedge \varphi_2 \wedge [(A\overline{\mathbf{U}^{\leq N}}B)) \vee (q\mathbf{S}^{\leq N}(B \vee a))]$: comme précédemement on peut considérer que le dernier bloc est mauvais.

On obtient une fonction de correction $\alpha(n) = (2n)^2 + n + 1$. On a encore un $+1$ dû au premier a qui cause une erreur locale.

4.i : $(q \vee (B\mathbf{R}A))\mathbf{S}^{\leq N}a$

Même chose que pour 3.ii sauf que le dernier B est autorisé à ne jamais apparaître, auquel cas A doit être toujours vrai dans le futur. φ_1 et φ_2 sont indentiques, et la formule 4.i est équivalente à $\varphi_1 \wedge \varphi_2 \wedge [(B\overline{\mathbf{R}}A)) \vee (q\mathbf{S}^{\leq N}(B \vee a))]$

5.i : $(q \vee (A\mathbf{U}^{\leq N}B))\mathbf{S}(a \wedge (A\mathbf{U}^{\leq N}B))$

Le premier B peut être avant ou après (au sens large) la position courante. S'il est avant, alors la situation est similaire à celle du cas 3.i : on pose $a' = B \wedge (A\mathbf{S}^{\leq N}a)$, et on doit vérifier $\psi = (q \vee (A\mathbf{U}^{\leq N}B))\mathbf{S}a'$, qui est exactement 3.i avec a' à la place de a. Puisque a' est une formule pur passé, la solution à 3.i en est aussi une pour ψ.

Si le premier B est à la position courante ou après, la situation est plus simple, on doit juste vérifier $\psi' = (A\mathbf{S}^{\leq N}a) \wedge (A\overline{\mathbf{U}^{\leq N}}B)$, qui est déjà une formule séparée. Finalement, 5.i est équivalente à $\psi \vee \psi'$.

5.ii : $(q \vee (A\mathbf{U}B))\mathbf{S}^{\leq N}(a \wedge (A\mathbf{U}B))$

Si on nomme $a' = B \wedge (A\mathbf{S}a)$, on doit vérifier ou bien $\psi = (q \vee (A\mathbf{U}B))\mathbf{S}^{\leq N}a'$, qui correspond au cas 3.ii, ou bien $\psi' = (A\mathbf{S}a) \wedge (A\overline{\mathbf{U}}B)$. Finalement 5.ii est équivalente à $\psi \vee \psi'$.

De la même façon, le cas 5.iii se réduit au cas 3.iii avec $a' = B \wedge (A\mathbf{S}^{\leq N}a)$, et $\psi' = (A\mathbf{S}^{\leq N}a) \wedge (A\overline{\mathbf{U}^{\leq N}}B)$.

Dans tous les cas, on peut prendre la fonction de correction $\alpha'(n) = \alpha(n) + n$, où $\alpha(n)$ vient du cas 3.x auquel on se ramène. La correction $+n$ est dûe aux erreurs commises par a'.

6.i : $(q \vee (B\mathbf{R}A))\mathbf{S}^{\leq N}(a \wedge (B\mathbf{R}A))$

Comme il a été fait dans le cas 4.i, il suffit de changer le Until du cas 5.ii en Release dans toutes les formules futures. On peut ensuite appliquer le cas 4.i avec $a' = B \wedge (A\mathbf{S}^{\leq N}a)$, et faire la conjonction avec $\psi' = (A\mathbf{S}^{\leq N}a) \wedge (A\overline{\mathbf{U}^{\leq N}}B)$.

On va expliciter la formule qu'on obtient dans ce cas : comme dans le cas 3.ii, soit

$$\begin{aligned}\varphi_1 &= [A\mathbf{S}(q\overline{\mathbf{S}^{\leq N}}(B \vee a'))]\mathbf{S}a',\\ \varphi_2 &= ([(q \vee B) \wedge \mathbf{Y}(q \vee B)] \vee [(A \vee B) \wedge (\mathbf{Y}(A \vee q \vee B))])\,\mathbf{S}^{\leq N}a'.\end{aligned}$$

Alors la formule 6.i est équivalente à $\varphi_1 \wedge \varphi_2 \wedge [(B\overline{\mathbf{R}}A)) \vee (q\mathbf{S}^{\leq N}(B \vee a'))]$.

Cas avec Q à la place de S

Finalement, remplacer **S** par **Q** n'est pas problématique. On peut utiliser les mêmes arguments pour séparer les formules : la seule différence est qu'on ne requière plus la présence d'un a dans le passé.

On peut noter que comme pour les opérateurs futurs, la variante quantitative de $B\mathbf{Q}A$ peut être décrite par $A\overline{\mathbf{S}^{\leq N}}(B\mathbf{Q}A)$, ce n'est donc pas nécessaire de le définir comme un opérateur de la syntaxe de $\mathrm{CLTL_P}$.

9.5.2 Terminaison de la procédure

On veut maintenant montrer qu'en réitérant ces diverses opérations, on finira toujours par obtenir une formule séparée, et ce pour toute formule de départ dans $\mathrm{CLTL_P}$.

Cette partie est une adaptation plus directe de [GHR94], peu de nouveaux éléments sont à prendre en considération.

L'idée principale pour que l'adaptation fonctionne est de considérer que $\mathbf{U}^{\leq N}$ and $\mathbf{R}$ ont un comportement similaire à $\mathbf{U}$ (et de même pour les opérateurs passés). Le principe de la preuve originale pourra ainsi être appliqué à chacun de ces opérateurs isolément.

Lemme 9.5.3 *Soit $\mathbf{U}'$ un opérateur de $\left\{\mathbf{R}, \mathbf{U}, \mathbf{U}^{\leq N}\right\}$. Soit A et B sont des formules de $CLTL_P$ sans opérateur temporel, et C et F des formules dont les seuls opérateurs futurs apparaissent sous le forme $A\mathbf{U}'B$, sans être imbriquées sous un opérateur passé. Soit $\mathbf{S}'$ un opérateur dans $\left\{\mathbf{Q}, \mathbf{S}, \mathbf{S}^{\leq N}\right\}$.*

Alors $C\mathbf{S}'F$ est équivalente à une formule séparée.

Démonstration Si $A\mathbf{U}'B$ n'apparaît pas, la formule est séparée. Sinon, un usage répété du Lemme 9.5.1 nous permet de réécrire $C\mathbf{S}F$ comme une combinaison booléenne de formule pur passé $C_1\mathbf{S}'C_2$, et de formules sous l'une des formes suivantes :

- $C_1\mathbf{S}'(C_2 \wedge (A\mathbf{U}'B))$
- $(C_1 \vee (A\mathbf{U}'B))\mathbf{S}'C_2$
- $(C_1 \vee (A\mathbf{U}'B))\mathbf{S}'(C_2 \vee (A\mathbf{U}''B))$

où C_1 et C_2 sont des formules pur passé. Ensuite les transformations précédentes (de 1.i à 6.i) nous permettent d'obtenir une combinaison booléenne de formules sous l'une des formes suivantes :

- $a\mathbf{Q}b$, $a\mathbf{S}b$, $a\mathbf{S}^{\leq N}b$, où a et b sont pur passé,
- C_1, C_2, A et B,
- $A\mathbf{U}'B$.

Ceci nous donne le résultat. □

On commence ensuite le procédé inductif consistant à enlever les opérateurs futurs $\mathbf{U}'$ imbriqués sous des opérateurs passés $\mathbf{S}'$, mais où les opérateurs futurs ne sont pas imbriqués entre eux.

Lemme 9.5.4 *On fixe de nouveau un $\mathbf{U}' \in \left\{\mathbf{R}, \mathbf{U}, \mathbf{U}^{\leq N}\right\}$. Soit A et B sans opérateurs temporels, et D une formule dont les seuls opérateurs futurs apparaissent comme $A\mathbf{U}'B$.*

Alors D est équivalente à une formule séparée, où les seuls opérateurs futurs sont toujours sous la forme $A\mathbf{U}'B$.

Démonstration Par récurrence sur le nombre maximal k de $\mathbf{S}'$ contenant un $A\mathbf{U}'B$.

Si $k = 0$ alors D est déjà séparée. Si $k > 0$, on applique le Lemme 9.5.3 à la sous-formule $C\mathbf{S}'F$ la plus profonde dans laquelle $A\mathbf{U}'B$ apparaît. On obtient

une formule équivalente où le paramètre d'induction est diminué. De plus, $A\mathbf{U}'B$ est toujours la seule occurence possible d'opérateurs futurs. □

Lemme 9.5.5 *Pour tout $i \in [1,n]$, soit A_i et B_i des formules de CLTL sans opérateurs temporels, et $\mathbf{U}_i \in \{\mathbf{R}, \mathbf{U}, \mathbf{U}^{\leq N}\}$. On suppose que les seuls occurences d'opérateurs futurs dans D sont les sous-formules $A_i\mathbf{U}_iB_i$. Alors D est équialent à une formule séparée.*

Démonstration On procède à une récurrence sur n.

Si $n = 1$, le Lemme 9.5.4 nous permet de conclure. Si $n > 1$ on va s'occuper seulement de $A_n\mathbf{U}_nB_n$. Pour ce faire, on remplace dans D chaque occurence de $A_i\mathbf{U}_iB_i$ par un nouvel atome q_i, ce qui nous donne une formule D'. En utilisant le Lemme 9.5.4, on obtient E' séparée équivalente à D', et utilisant les atomes $(q_i)_{1\leq i\leq n-1}$. Les seuls occurences d'opérateurs futurs de E' sont les $A_n\mathbf{U}_nB_n$.

E' iest une combinaison booléenne de formules pur présent, pur futur $A_n\mathbf{U}_nB_n$, et pur passé D_j avec atomes $(q_i)_{1\leq i\leq n-1}$ en plus des normaux. On peut maintenant remplacer dans tous les D_j chaque q_i par $A_i\mathbf{U}_iB_i$. Par induction chaque D_j est équivalent à une formule séparée, ce qui nous permet finalement d'obtenir la formule séparée désirée.

□

La prochaine étape est d'autorise l'imbrication de plusieurs $\mathbf{U}'$ sous des $\mathbf{S}'$, mais pas encore de $\mathbf{S}'$ sous des $\mathbf{U}'$

Lemme 9.5.6 *Soit D une formule de $CLTL_P$ sans $\mathbf{S}'$ sous un $\mathbf{U}'$. Alors D est équivalente à une formule séparée.*

Démonstration Par récurrence sur la profondeur maximale n d'imbrication de $\mathbf{U}'$ sous un $\mathbf{S}'$.

Le Lemme 9.5.5 règle le cas $n = 1$.

On suppose $n > 1$, soi $A_i\mathbf{U}_iB_i$ pour $i \in [1,k]$ toutes les sous-formules de D faisant intervenir un opérateur futur $\mathbf{U}_i \in \{\mathbf{R}, \mathbf{U}, \mathbf{U}^{\leq N}\}$ qui n'est pas sous un autre opérateur futur. Chaque A_i et B_i est une combinaison booléenne d'atomes et de formules $X_{ij}\mathbf{U}_{ij}Y_{ij}$. On remplace chacune de ces formules par un nouvel atome x_{ij}, de façon à obtenir des formules A'_i et B'_i.

On remplacer maintenant dans D chaque $A_i\mathbf{U}_iB_i$ par $A'_i\mathbf{U}_iB'_i$, ce qui nous donne une formule D'. Par le Lemme 9.5.5, D' est équivalente à une formule séparatée E' avec atomes x_{ij}. En remplaçant les x_{ij} par $X_{ij}\mathbf{U}_{ij}Y_{ij}$ dans E', on obtient une formule équivalente à l'originale, où le paramètre d'induction est diminuée, que l'on peut séparer par hypothèse de récurrence.

□

On peut maintenant montrer le Théorème de Séparation, c'est-à-dire transformer toute formule D de $\mathrm{CLTL_P}$ en une formule équivalente séparée.

On va réutiliser les notations de [GHR94] : si A est une formule de $\mathrm{CLTL_P}$, on appelle la *profondeur d'alternation* de A le nombre maximal d'alternations entre les opérateurs passés et futurs imbriqués dans A.

On procède par induction sur la profondeur d'alternation de la formule D à séparer.

Si elle vaut zéro ou un, alors D est déjà séparée. On suppose donc qu'elle vaut au moins deux.

D est une combinaison booléenne d'atomes et de formules de la forme $D_1 \mathbf{S}' D_2$, $D_1 \mathbf{U}' D_2$. Par symétrie, il suffit de montrer le résultat pour les formules de la forme $D = D_1 \mathbf{S}' D_2$.

Soit $A_i \mathbf{U}_i B_i$ pour $i \in [1, k]$ les sous-formules couvrant les apparances maximales (pour l'imbrication) des opérateurs futurs dans D.

On remplace dans D chaque sous-formule $E_{ij} \mathbf{S}_{ij} F_{ij}$ apparaissant dans un certain $A_i \mathbf{U}_i B_i$ par un nouvel atome x_{ij}. On obtient ainsi une formule satisfaisant les hypothèses du Lemme 9.5.6, que l'on peut donc séparer en une formule E'. En remplaçant chaque x_{ij} par $E_{ij} \mathbf{S}_{ij} F_{ij}$ dans E', on obtient une nouvelle formule où la profondeur d'alternation est diminuée. L'hypothèse de récurrence nous permet de conclure.

On peut remarquer que chaque pas de récurrence peut introduire une fonction de correction quadratique. On obtient donc une fonction de correction globale de la forme $\alpha(n) = O(n^{2^{nd(D)}})$, où $nd(D)$ est la profondeur de D.

9.6 Lien avec les B-VWAA

Théorème 9.6.1 ([KV12]) *Il est équivalent pour une fonction de coût régulière d'être reconnue par*

- *un B-VWAA,*
- *un B-VWAA à un seul compteur,*
- *une formule de CFO,*
- *une formule de CLTL.*

On décrit ici brièvement les idées nécessaires pour prouver l'équivalence entre CLTL et les B-VWAA (à un seul compteur).

Le sens le plus simple est la transformation d'une formule de CLTL θ en un B-VWAA à un compteur.

On définit le B-VWAA $\mathcal{A}_\theta = \langle Q, \mathbb{A}, q_0, F, \{\gamma\}, \delta \rangle$ comme suit :

- $Q := \{\varphi : \varphi \text{ est une sous-formule de } \theta\}$
- $q_0 := \theta$
- $\delta : Q \times \mathbb{A} \to \mathcal{B}^+(\mathbb{C} \times Q)$ est défini par

$$\delta(a, b) := \begin{cases} \mathsf{true} & \text{si } b = a \\ \mathsf{false} & \text{sinon} \end{cases}$$

$$\delta(\varphi \vee \psi, b) := \delta(\varphi, b) \vee \delta(\psi, b)$$

$$\delta(\varphi \wedge \psi, b) := \delta(\varphi, b) \wedge \delta(\psi, b)$$

$$\delta(\varphi \mathbf{U}^{\leq N} \psi, b) := (\mathtt{r}, \psi) \vee (\mathtt{ic}_j, \varphi \mathbf{U}^{\leq N} \psi) \vee ((\mathtt{r}, \varphi) \wedge (\varepsilon, \varphi \mathbf{U}^{\leq N} \psi))$$

$$\delta(\varphi \mathbf{U} \psi, b) := (\varepsilon, \psi) \vee ((\varepsilon, \varphi) \wedge (\varepsilon, \varphi \mathbf{U} \psi))$$

$$\delta(\varphi \mathbf{R} \psi, b) := (\varepsilon, \psi) \wedge ((\varepsilon, \varphi) \vee (\varepsilon, \varphi \mathbf{R} \psi))$$

– $F := \{\varphi : \varphi \text{ est de la forme} A\mathbf{R}B\}$

L'automate $\mathcal{A}_\theta$ est bien un B-VWAA, et il ne comporte qu'un seul compteur. La preuve de correction de cet automate est dûe à Michael Vanden Boom, et peut se trouver dans [KV12].

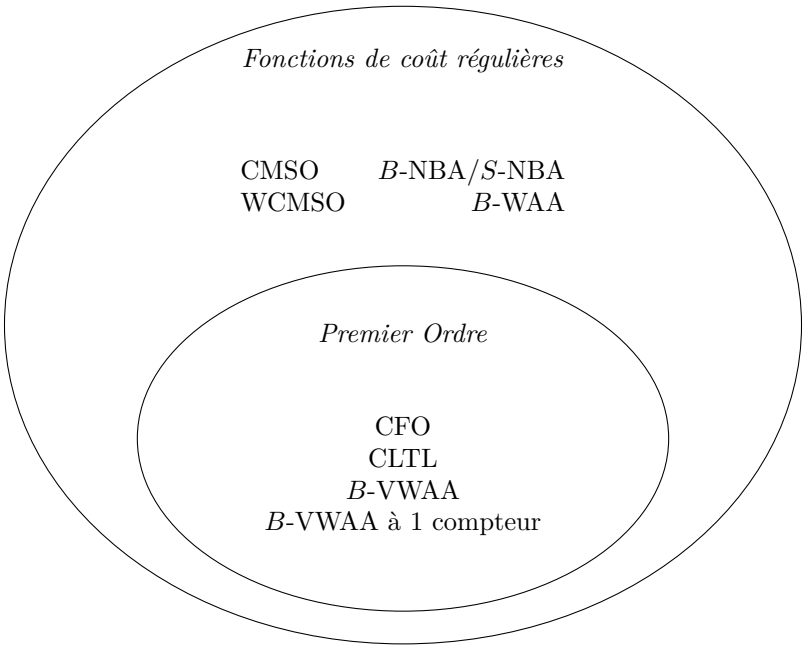
La réciproque est plus difficile. Il faut en effet partir d'un B-VWAA $\mathcal{A}$ comportant un nombre arbitraire de compteurs, et construire une formule CLTL $\varphi_\mathcal{A}$ équivalente à $\mathcal{A}$. Cette construction est dûe à Michael Vanden Boom.

Il faut d'abord transformer $\mathcal{A}$ en un autre B-VWAA $\mathcal{A}'$ équivalent, respectant des contraintes additionnelles. On peut ensuite décrire les exécutions acceptantes de $\mathcal{A}'$ au moyen d'une formule de CLTL. Cette formule utilisera des conjonctions et des disjonctions pour refléter celles de δ, et des opérateurs $\mathbf{U}^{\leq N}$ pour garantir une valeur bornée des compteurs. On peut se reporter à [KV12] pour le détail de la construction.

Chapitre 10

Conclusion

On a montré que les théorèmes d'équivalence entre les logiques FO et LTL d'une part, et MSO et WMSO d'autre part, s'étendent des langages classiques aux fonctions de coût. Toutes les preuves classiques n'étaient pas directement généralisables, à cause de plusieurs phénomènes. En effet, certaines utilisent l'existence d'un modèle d'automate déterministe calculant tout langage régulier, tandis que pour d'autres, l'intéraction entre la condition d'acceptation et le comportement des compteurs est problématique. En choisissant les bonnes preuves à généraliser, et en étudiant la structure spécifique des automates de coût, on a pu obtenir les équivalences voulues, résumées dans le diagramme suivant :



Remarquons que ces équivalences sont valides en particulier sur les mots finis. La plupart deviennent alors triviales, mais la généralisation du théorème de Kamp est tout aussi difficile sur mots finis. On peut donc compléter le schéma récapitulatif de la partie sur les mots finis, grâce au corollaire suivant :

Corollaire 10.0.2 *Soit f une fonction de coût régulière sur mots finis. Alors les assertions suivantes sont équivalentes :*

- *Le semigroupe de stabilisation de f est apériodique,*
- *f est reconnue par une formule de CLTL,*
- *f est reconnue par une formule de CFO,*

On a montré que de nombreux théorèmes concernant les langages réguliers sur mots infinis sont toujours valides dans le cadre des fonctions de coût régulières. Ceci montre d'une part que chaque notion a été étendue de manière cohérente, et d'autre part que cette théorie constitue une généralisation robuste, qui peut encore bénéficier de nombreux outils développés initialement pour les langages. L'absence de modèle d'automate déterministe est un obstacle qu'il faut parfois contourner, mais la multitude des approches existantes nous a permis de le faire ici avec succès.

Troisième partie

Fonctions de coût sur arbres infinis

Chapitre 11

Fonctions et automates sur arbres infinis

On veut maintenant calculer des fonctions sur les arbres infinis étiquetés par un alphabet $\mathbb{A}$.

Le modèle d'arbre infini généralise celui de mot infini, et permet de modéliser le comportement de systèmes branchants. On peut par exemple utiliser le branchement pour exprimer les différents comportements possibles d'un environnement, ou d'un adversaire. Typiquement, dans un tel contexte, on exige du système de produire un comportement désirable dans tous les cas de figures possibles : ceci sera formalisé par des automates dont les conditions d'acceptations doivent être vérifiées sur toutes les branches de l'arbre lu.

La théorie des langages réguliers sur arbres infinis s'est montré très fructueuse, on peut notamment cite le célèbre théorème de Rabin établissant la décidabilité de MSO sur les arbres infinis, au moyen d'un nouveau genre d'automates [Rab69]. Beaucoup de phénomènes nouveaux apparaissent lorsque l'on passe au modèle des arbres infinis, et les outils à mettre en oeuvre se diversifient : les aspects topologiques des langages prennent par exemple une importance nouvelle.

Pour simplifier les notations, on va ici se restreindre aux arbres binaires complets étiquetés par un alphabet fini $\mathbb{A}$. Gardons à l'esprit que tout arbre de degré borné peut s'encoder au moyen d'un arbre binaire complet, et que les résultats qu'on obtient ici (notamment en termes d'expressivité) peuvent s'étendre sur la classe des arbres de degré quelconque.

11.1 Fonctions de coût sur arbres infinis

Formellement, l'arbre binaire complet peut être défini comme l'ensemble $\mathcal{T} := \{0,1\}^*$. En effet, on peut considérer que la lettre 0 permet d'indiquer le fils gauche, et 1 le fils droit. En lisant un mot de $\{0,1\}^*$ de gauche à droite, on peut donc suivre le chemin correspondant dans l'arbre binaire complet. Ainsi, chaque

mot de $\{0,1\}^*$ décrira un noeud de l'arbre (celui au bout du chemin décrit), et en particulier le mot vide ϵ désignera la racine.

On note $\mathcal{T}_{\mathbb{A}}$ l'ensemble des arbres binaires infinis complets étiquetés par $\mathbb{A}$. Formellement, $\mathcal{T}_{\mathbb{A}}$ est l'ensemble des fonctions de $\mathcal{T}$ vers $\mathbb{A}$, qui étiquettent chaque noeud de l'arbre binaire complet.

On veut maintenant calculer des fonctions de coût non plus sur les mots sur un alphabet $\mathbb{A}$, mais sur ces arbres. Les fonctions de coûts considérées dans la suite seront donc représentées par des fonctions $\mathcal{T}_{\mathbb{A}} \to \mathbb{N}_\infty$.

Si x est un noeud, les *fils* de x sont les noeuds $x0$ et $x1$ qui sont situés immédiatement dessous. Les *descendants de* x sont tous les noeuds de la forme $x\{0,1\}^*$, qui sont situés dessous au sens large,.

On utilisera l'ordre préfixe $\sqsubseteq$ sur les noeuds de $\{0,1\}^*$: $x \sqsubseteq y$ si x est un préfixe de y, autrement dit s'il existe v tel que $xv = y$.

Un ensemble $L \subseteq \mathcal{T}$ est *clos par préfixe* si pour tous $x \in L$ et $y \sqsubseteq x$, on a $y \in L$.

Soit $x, y \in \mathcal{T}$, on note $x \wedge y$ le plus grand (pour $\sqsubseteq$) z tel que $z \sqsubseteq x$ et $z \sqsubseteq y$.

Une *branche* de $\mathcal{T}$ est un ensemble de noeuds $\pi \subseteq \{0,1\}^*$ clos par préfixe tel que pour tout $n \in \mathbb{N}$, $\pi \cap \{0,1\}^n$ est un singleton.

11.1.1 Un exemple détaillé

Soit $\mathbb{A} = \{a, b\}$ et f la fonction de coût qui à t associe le nombre de a dans t. Soit g la fonction qui à t associe le nombre maximum de a sur une branche de t.

Soit π une branche de $\mathcal{T}$, on dit qu'un ensemble de noeuds $P \subseteq \mathcal{T}$ est un *peigne* de squelette π si P contient π, est clos par $\sqsubseteq$ et pour tous $x, y \in P$, $x \wedge y \in \pi$. Un peigne est donc constitué d'une branche infinie, et de branches infinies qui en partent mais ne se subdivisent plus.

Soit h la fonction qui à t associe le nombre maximum de a sur un peigne de t.

Alors $f \not\approx g$ mais $f \approx h$.

Démonstration Pour montrer que $f \not\approx g$, on peut définir l'arbre t qui comporte des a sur les noeuds de 0^*1 et des b partout ailleurs. On a $f(t) = \infty$ mais $g(b) = 1$, ce qui suffit à montrer que $f \not\approx g$.

On a trivialement $h \leq f$, il reste à montrer l'existence d'une fonction de correction α telle que $f \preccurlyeq_\alpha h$.

On montre d'abord que si $f(t) = \infty$ alors $h(t) = \infty$. Pour ce faire, étant donné un arbre t contenant une infinité de a, on définit un peigne P qui en contient aussi une infinité.

On construit une branche π par récurrence :$\varepsilon \in \pi$, et pour tout $x \in \pi$, on choisit son successeur y comme étant un noeud ayant une infinité de a parmi ses descendants. Pour l'autre fils z, on choisit une branche π_z qui contient au moins un a, si c'est possible. L'union de π et des π_z forme un peigne. Il est facile de voir que ce peigne contient une infinité de a : s'ils ne sont pas sur π, alors ils seront sur les π_z.

On suppose maintenant que $f(t)$ est fini.

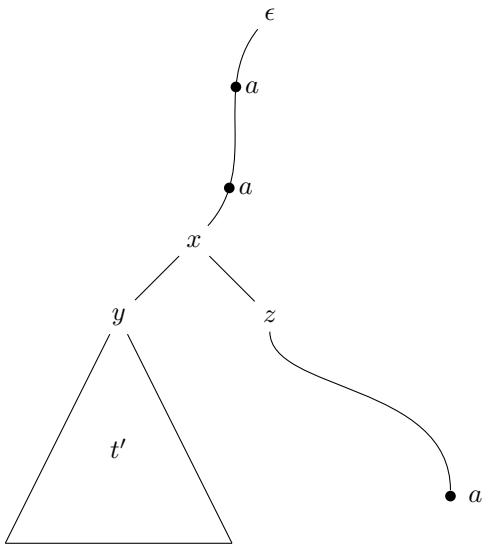
On note $|P|_a$ le nombre de a dans un ensemble de noeuds P, s'il n'y a pas d'ambiguité sur l'étiquetage de ces noeuds (un arbre t est fixé).

On procède par récurrence sur $f(t)$ Le résultat que l'on veut montrer est que pour tout t, il existe un peigne $P(t)$ tel que $|P(t)|_a \geq \log_2(f(t))$. Si $f(t) = 0$ le résultat est trivial.

Soit t tel que $f(t) > 0$, on suppose le résultat vrai pour $f(t') < f(t)$.

Si tous les a sont situés sur une même branche π, alors π constitue un peigne, et $|\pi|_a = f(t) \geq \log_2(f(t))$.

Sinon, soit x le plus petit noeud (pour $\sqsubseteq$) dont les deux fils ont un descendant étiqueté par a. Soit y le fils de x ayant le plus de tels descendants, et z le second fils de x. Soit t' l'arbre enraciné en y.



Soit $n \geq 0$ le nombre de a sur des sommets $x' \sqsubseteq x$. On a $\frac{f(t)-n}{2} \leq f(t') < f(t) - n$, donc par hypothèse de récurrence, il existe un peigne P' sur t' avec $|P'|_a \geq \log_2(f(t'))$. On obtient donc $|P'|_a \geq \log_2(f(t) - n) - 1$. Soit P le peigne contenant P', ainsi que les sommets $x' \sqsubseteq x$, et une branche contenant z et un de ses descendants étiqueté a. Alors $|P|_a \geq |P'|_a + n + 1 \geq \log_2(f(t) - n) + n \geq \log_2(f(t))$.

Par récurrence, on obtient donc $f \preccurlyeq_\alpha h$, avec $\alpha(n) = 2^n$. □

11.2 Automates de coût sur arbres infinis

11.2.1 B- et S-valuations

On reprend les valuations val_B et val_S définies sur mots infinis, et on les étend aux arbres infinis. Notons qu'on les utilisera parfois encore sur mots infinis (par exemple sur des branches).

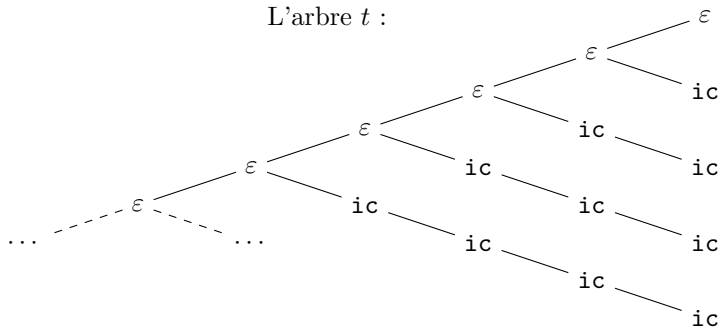
On garde les deux alphabets d'actions atomiques $\mathbb{C}_B = \{\mathtt{ic}, \varepsilon, \mathtt{r}\}^\Gamma$ et $\mathbb{C}_S = \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^\Gamma$.

On se fixe un ensemble Γ de compteurs. Soit $\mathbb{C} = \mathbb{C}_B^\Gamma$ et $t \in \mathcal{T}_\mathbb{C}$, on définit $val_B(t) = \sup\{val_B(\pi) : \pi \text{ branche de } t\}$.

Soit $\mathbb{C} = \mathbb{C}_S^\Gamma$ et $t \in \mathcal{T}_\mathbb{C}$, on définit $val_S(t) = \inf\{val_S(\pi) : \pi \text{ branche de } t\}$.

Autrement dit, dans les deux cas, la condition imposée par les compteurs doit être vérifiée sur toutes les branches.

Par exemple soit t est l'arbre suivant étiqueté par $\mathbb{C}_B$ (les noeuds non représentés sont étiquetés ε) :



Alors $val_B(t) = \infty$: même si toutes les branches ont une valeur finie, leur supremum est infini.

11.2.2 Automates de coût alternants sur arbres infinis

On va donner ici directement la définition de la classe générale des B- (resp. S-) automates de Büchi alternants sur arbres infinis. Les diverses classes de B- (resp. S-)automates sur arbres infinis seront ensuite définies par des restrictions sur cette classe générale.

Un *B-automate de Büchi alternant*, abrégé B-ABA$\mathcal{A} = \langle Q, \mathbb{A}, In, F, \Gamma, \delta\rangle$ (resp. S-automate S-ABA) sur arbres infinis possède

- un ensemble fini d'états Q,
- un alphabet fini $\mathbb{A}$,
- un ensemble d'états initiaux $In \subseteq Q$,
- un ensemble d'états de Büchi $F \subseteq Q$,
- un ensemble fini Γ de B-compteurs (resp. S-compteurs). Soit $\mathbb{C} := \mathbb{C}_B^\Gamma$ (resp. $\mathbb{C}_S^\Gamma$) l'ensemble des actions atomiques sur Γ.
- une fonction de transition $\delta : Q \times \mathbb{A} \to \mathcal{B}^+(\{0,1\} \times \mathbb{C} \times Q)$, où $\mathcal{B}^+(\{0,1\} \times \mathbb{C} \times Q)$ est l'ensemble des combinaisons booléennes positives (sans négation) d'éléments de $(\mathbb{C} \times Q\,\{0,1\})$. Pour tout $(q,a) \in Q \times \mathbb{A}$, on peut écrire $\delta(q,a)$ comme une disjonction de clauses, chaque clause étant une conjonction d'éléments $(d_i, \nu_i, q_i) \in \{0,1\} \times \mathbb{C} \times Q$. La nouvelle composante $\{0,1\}$ sert à préciser quelle est la branche de l'arbre t lu en entrée qui sera suivie par l'automate.

Le comportement d'un tel automate $\mathcal{A}$ sur un arbre t peut être vu comme un jeu $(\mathcal{A}, t)$ entre deux joueurs : Eve et Adam. A chaque position q dans l'automate, si a est la lettre lue, Eve choisit une clause dans $\delta(q, a)$, puis Adam choisit un élément (d, ν, q) dans cette clause, qui déterminera l'action effectuée et le nouvel état de l'automate. Dans ce formalisme, seule une branche de l'arbre de l'arbre d'entrée, fixée par les choix des joueurs, est lue par l'automate.

Le comportement de l'automate est ensuite le même que sur les mots infinis sur cette branche particulière : le but d'Eve est d'accepter le mot d'entrée avec la plus petite (resp. grande pour les S-automates) valuation possible pour l'action construite, et celui d'Adamest de l'en empêcher.

Une *exécution* de $\mathcal{A}$ sur un arbre $t \in \mathcal{T}_{\mathbb{A}}$ décrit une suite de transitions que peut emprunter $\mathcal{A}$ sur t. Elle est représentée par une suite $q_0, (d_1, \nu_1, q_1), (d_2, \nu_2, q_2), \ldots$ compatible avec t et $\delta : q_0$ est initial, et pour tout $i \in \mathbb{N}$, $(d_{i+1}, \nu_{i+1}, q_{i+1})$ apparaît dans $\delta(q_i, t(d_1 \ldots d_i))$.

Une *stratégie* pour Eve (resp. Adam) dans le jeu $(\mathcal{A}, t)$ est une fonction qui fixe le prochain choix d'Eve (resp. d'Adam),en se basant sur l'historique de l'exécution (resp. l'histoire de l'exécution et le choix de disjunction d'Eve).

Une stratégie est *à mémoire finie* m si le nombre d'états de mémoire nécessaires pour choisir le prochain coup est fini, et vaut au plus m. Une stratégie est *positionnelle* si elle est à mémoire finie 0, c'esst-à-dire que le joueur a seulement besoin de connaître la position courante dans le jeu pour décider de son prochain coup.

On peut remarquer que choisir une stratégie pour Eve et Adam ainsi qu'un arbre t fixe une exécution de $\mathcal{A}$ sur t.

On dit qu'une exécution est *compatible* avec une stratégie σ pour Eve s'il existe une stratégie σ' pour Adam telle que π est l'exécution générée par σ et σ'.

Une exécution π est *acceptante* s'il existe $q \in F$ apparaissant infiniment souvent sur π (c'est-à-dire π satisfait la condition de Büchi). On définit $val_B(\pi)$ (resp. $val_S(\pi)$) en prenant simplement la projection de π sur $\mathbb{C}$, et en l'évaluant comme dans le cas des mots infinis. On pourra considérer que si π ne satisfait pas la condition de Büchi, alors $val_B(\pi) = \infty$ (resp. $val_S(\pi) = 0$). Si $n \in \mathbb{N}$, une n-exécution d'un B-ABA (resp. S-ABA) est une exécution π telle que $val_B(\pi) \leq n$ (resp. $val_S(\pi) \geq n$).

On associe une valeur a une statégie σ pour Eve dans un B-automate (resp. S-automate) en posant $val_B(\sigma) := \sup\{val(\pi) : \pi \text{ est compatible avec } \sigma\}$, resp. $val_S(\sigma) := \inf\{val(\pi) : \pi \text{ est compatible avec } \sigma\}$. Si $n \in \mathbb{N}$, une n-stratégie pour Eve dans un B-ABA (resp. S-ABA) est une stratégie σ telle que $val_B(\sigma) \leq n$ (resp. $val_S(\sigma) \geq n$).

Finalement, si $\mathcal{A}$ est un B-ABA et $\mathcal{A}'$ est un S-ABA sur arbres infinis, on pose

$$[\![\mathcal{A}]\!]_B(t) := \inf\{val_B(\sigma) : \sigma \text{ est une stratégie d'Eve dans } (\mathcal{A}, t)\}$$
$$[\![\mathcal{A}]\!]_S(t) := \sup\{val_S(\sigma) : \sigma \text{ est une stratégie d'Eve dans } (\mathcal{A}, t)\}$$

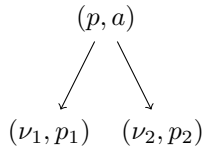
$[\![\mathcal{A}]\!]_B$ et $[\![\mathcal{A}']\!]_S$ seront considérées comme des fonctions de coût, donc on les

évaluera toujours modulo $\approx$. Si f est une fonctio $\mathcal{T}_{\mathbb{A}} \to \mathbb{N}_\infty$ et que $\mathcal{A}$ est un automate tel que $f \approx [\![\mathcal{A}]\!]$, on dira que $\mathcal{A}$ *reconnaît* la fonction de coût f.

11.2.3 Automates non-déterministes

Si pour tout $(q, a) \in Q \times \mathbb{A}$, $\delta(q, a)$ est de la forme $\bigvee_i (0, \nu_i, q_i) \wedge (1, \nu_i', q_i')$, on dit que l'automate est *non-déterministe*. En effet, on ne voit plus cet automate comme définissant un jeu, mais plutôt comme étant dirigé seulement par Eve, sur toutes les branches de manière simultanée.

Comme dans le cas des mots infinis, on pourra présenter δ non plus comme une fonction mais comme un ensemble de transitions, noté alors Δ. On aura alors $\Delta \subseteq Q \times \mathbb{A} \times (\mathbb{C} \times Q)^2$. Chaque élément $(p, a, \nu_1, q_1, \nu_2, q_2)$ de δ représentera une alternative $(0, \nu_1, q_1) \wedge (1, \nu_2, q_2)$ de $\delta(p, a)$.



Dans ce cas, on dira plutôt qu'une exécution est un arbre R étiqueté par $Q \times \mathbb{C}$, correspondant aux transitions choisies dans Δ. Une exécution R est acceptante si elle l'est sur toutes les branches, et sa valeur est $val_B(R)$ tel que définie dans la Section 11.2.1. Cela correspond à prendre le supremum de la valeur des branches pour les B-automates, et l'infimum pour les S-automates.

On notera B-NBA pour les B-automates de Büchi non-déterministes, et S-NBA pour les S-automates de Büchi non-déterministes.

11.2.4 B- et S-automates faibles

Ces automates généralisent les automates faibles alternants classiques, introduits par Muller et al. dans [MSS92]. Cette généralisation aux automates de coût est introduite par Vanden Boom dans [VB11].

De la même manière que sur les mots infinis, un B-ABA $\mathcal{A} = \langle Q, \mathbb{A}, In, F, \Gamma, \delta \rangle$ sur arbres infinis est un *B- (ou S-) automate faible* (B-WAA, S-WAA) si son ensemble d'états Q peut être partitionné en des ensembles $Q_1, \ldots, Q_k$, ordonnés partiellement par $\leq$, tels que :

- pour tout $q, q' \in Q_i$, $q \in F$ si et seulement si $q' \in F$;
- si $q_j \in Q_j$ peut être atteint depuis $q_i \in Q_i$ par des transitions de δ, alors $Q_j \leq Q_i$.

Cela revient à interdire l'existence d'un cycle comportant simultanément des états de Büchi (acceptants) et des états non Büchi (rejetants). Toute exécution de $\mathcal{A}$ va donc se stabiliser soit dans des états acceptants, soit dans des états rejetants.

Si π est une exécution (le long d'une branche de l'arbre d'entrée), on définit $alt(\pi)$ le nombre d'alternations entre états acceptants et refusants dans π.

La définition des automates faibles revient à imposer l'existence d'une borne k telle que pour toute exécution π, $alt(\pi) \leq k$.

Remarquons que cette définition est exactement la même que dans le cas classique, et ignore les actions de compteurs dans les transitions de l'automate.

11.2.5 Exemples

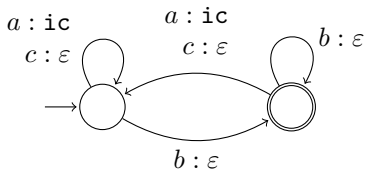
Soit $\mathbb{A} = \{a, b, c\}$ et f la fonction de coût sur $\mathcal{T}_{\mathbb{A}}$ définie par

$$f(t) = \begin{cases} \infty \text{ s'il y a une branche de } t \text{ contenant un nombre fini de } b \\ \sup\{|\pi|_a : \pi \text{ branche de } t\} \text{ sinon} \end{cases}$$

Comme d'habitude $|\pi|_a$ désigne le nombre de a dans π, possiblement ∞.

Ona va définir un B-NBA $\mathcal{U}$, un S-NBA $\mathcal{U}'$, et un B-WAA $\mathcal{A}$, tels que $f \approx [\![\mathcal{U}]\!] \approx [\![\mathcal{U}']\!] \approx [\![\mathcal{A}]\!]$.

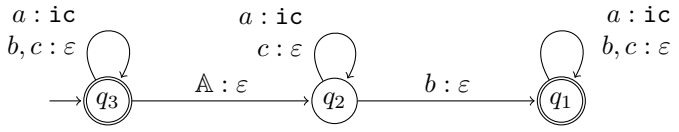
Le principe de $\mathcal{U}$ est de compter a tout en vérifiant qu'il y a une infinité de b. Sur les mots infinis, cela peut être effectué par un B-automate de Büchi déterministe $\mathcal{B}$. Il suffit donc de simuler $\mathcal{B}$ simultanément sur toutes les branches de l'arbre d'entrée pour obtenir $\mathcal{U}$. L'automate $\mathcal{B}$ est représenté ci-dessous, les états de Büchi étant marqués par des doubles cercles.



Au contraire, l'automate $\mathcal{U}' = \langle \{p_a, p_b, q_b, \top\}, \mathbb{A}, \{p_a, p_b\}, \{q_b, \top\}, \{\gamma\}, \delta \rangle$ va tenter de trouver une branche π qui comporte soit beaucoup de a (état p_a), soit un nombre fini de b (état p_b), de manière à prouver que la valeur de f est grande (∞ dans le second cas). L'état q_b sera utilisé quand Eve a deviné la position du dernier b, et doit donc vérifier qu'il n'y a plus de b sur la branche π choisie. L'état $\top$ signifie que le reste de la branche n'a plus d'importance. La table de transition δ pour $\mathcal{U}'$ est décrite ci-dessous. Pour la lisibilité, on notera $(\nu_1, q_1) \wedge (\nu_2, q_2)$ au lieu de $(0, \nu_1, q_1) \wedge (1, \nu_2, q_2)$ dans la table de transition.

δ	p_a	p_b	q_b	$\top$
a	$((\mathtt{i}, p_a) \wedge (\varepsilon, \top))$ $\vee((\varepsilon, \top) \wedge (\mathtt{i}, p_a))$ $\vee((\mathtt{cr}, \top) \wedge (\varepsilon, \top))$ $\vee((\varepsilon, \top) \wedge (\mathtt{cr}, \top))$	$((p_b) \wedge (\varepsilon, \top))$ $\vee((\varepsilon, \top) \wedge (\varepsilon, p_b))$ $\vee((\varepsilon, q_b) \wedge (\varepsilon, \top))$ $\vee((\varepsilon, \top) \wedge (\varepsilon, q_b))$	$((\varepsilon, q_b) \wedge (\varepsilon, \top))$ $\vee((\varepsilon, \top) \wedge (\varepsilon, q_b))$	$(\varepsilon, \top) \wedge (\varepsilon, \top)$
b	$((\varepsilon, p_a) \wedge (\varepsilon, \top))$ $\vee((\varepsilon, \top) \wedge (\varepsilon, p_a))$ $\vee((\mathtt{cr}, \top) \wedge (\varepsilon, \top))$ $\vee((\varepsilon, \top) \wedge (\mathtt{cr}, \top))$	$= \delta(p_b, a)$	*false* (disjonction vide)	$(\varepsilon, \top) \wedge (\varepsilon, \top)$
c	$= \delta(p_a, b)$	$= \delta(p_b, a)$	$= \delta(q_b, a)$	$(\varepsilon, \top) \wedge (\varepsilon, \top)$

Finalement, $\mathcal{A}$ va être construit de manière à ce qu'Adam ait le contrôle complet des transitions. Son rôle va être de sélectionner une branche, et simuler l'automate suivant (dont il contrôle le non-déterminisme) sur cette branche :



S'il y a une branche π avec un nombre de b fini, Adam peut choisir π et stabiliser dans l'état q_2, en prenant la transition de q_1 à q_2 après le dernier b. Ceci témoigne d'une valeur ∞ pour f. Sinon, Adam peut choisir la branche comportant le plus de a (ou un nombre de a arbitrairement grand si cette branche n'est pas définie). Il est clair que cet automate est faible : les seuls cycles sont triviaux.

11.2.6 B-automates quasi-faibles

On veut définir une extension des B-WAA, qui préserve la propriété selon laquelle toute exécution acceptante se stabilise soit dans des états acceptants, soit dans des états refusants (on n'autorise pas une infinité d'alternations).

L'idée des B-WAA (et S-WAA) est d'expliciter une borne sur le nombre d'alternations, directement lisible dans la structure de l'automate. Ici, dans le cas des B-automates, l'on dispose d'un nouvel outil pour définir une telle borne : les compteurs de l'automate.

Dans un B-automate, pour $n \in \mathbb{N}$, une n-exécution doit effectuer au plus n incréments entre deux resets consécutifs. Cette valeur n'est pas connue a priori par l'automate, on doit utiliser les compteurs et non les états pour garantir cette condition.

Si l'on peut garantir l'existence d'une fonction de correction α telle que pour toute n-exécution π, $alt(\pi) \leq \alpha(n)$, on obtiendra de nouveau la propriété désirée. Il devient quand même possible d'avoir une exécution π avec une infinité d'alternations, mais dans ce cas là $val_B(\pi) = \infty$, donc ces exécutions ne permettent pas à Eve d'accepter l'arbre d'entrée avec une valeur finie.

Définition 11.2.1 *Soit $\mathcal{A} = \langle Q, \mathbb{A}, In, F, \Gamma, \delta \rangle$ un B-ABA sur arbres infinis. On dit que $\mathcal{A}$ est un* B-automate quasi-faible *(B-QWAA) s'il existe une fonction de correction α telle que pour toute exécution π de $\mathcal{A}$, $alt(\pi) \leq \alpha(val_B(\pi))$.*

Remarque 11.2.2 *Tout B-WAA est en particulier un B-QWAA, puisqu'il suffit de choisir $\alpha(n) = k$ pour tout n, où k est la borne donnée par le B-WAA.*

Remarque 11.2.3 *Si l'on change la condition d'acceptation des B-WAA et des B-QWAA de Büchi à coBüchi, cela ne change pas la fonction de coût définie par ces automates. En effet, dans un B-WAA aucune exécution ne changerait de statut (acceptante ou refusante). Quant aux B-QWAA, les seules exécutions*

qui changeraient de statut seraient de valeur ∞, et se comporteraient donc tous les cas comme des exécutions refusantes.

On donne également une caractérisation structurelle des B-QWAA :

Proposition 11.2.4 *Un B-ABA $\mathcal{A}$ est un B-QWAA si et seulement si dans tout cycle de transitions accessible depuis un état initial, et contenant des états acceptants et refusants, il existe un compteur qui est incrémenté mais qui n'est pas remis à zéro.*

Démonstration

Cette caractérisation est intuitive, puisqu'elle impose de « compter » les cycles qui contiennent des états acceptants et refusants. Compter les cycles est suffisant, car le long d'une exécution, le nombre d'alternations qui ne se trouvent pas dans un cycle est borné par la structure de l'automate.

Il faut cependant garantir que la possibilité de jouer sur différents compteurs ne permet pas de « tricher », et donc prouver formellement que cette caractérisation est bien équivalente à la définition des B-QWAA.

La caractérisation donnée par l'énoncé sera appelée *condition de cycle*, abrégée CC. Soit $\mathcal{A} = \langle Q, \mathbb{A}, In, F, \Gamma, \delta \rangle$ un B-ABA.

Quasi-faible $\implies$ CC

Supposons que $\mathcal{A}$ ne satisfait pas CC, c'est-à-dire que $\mathcal{A}$ contient un cycle c accessible, contenant des états dans F et d'autres dans son complément, et tel que pour tout $\gamma \in \Gamma$, si c contient $\mathtt{ic}_\gamma$ il contient également $\mathtt{r}_\gamma$.

Puisque c est accessible, il existe un arbre $t \in \mathcal{T}_\mathbb{A}$ et une exécution π de $\mathcal{A}$ sur t qui atteint c au bout d'un préfixe fini u de π, et qui répète ensuite c ad infinitum : $\pi = uc^\omega$. Notons qu'en particulier, la branche de t correspondant à π doit être ultimement périodique.

L'exécution π est acceptante, puisque c contient des états de F. Mais on a $alt(\pi) = \infty$, et $val_B(\pi) \leq val(u) + 2val_B(c)$, puisque les compteurs incrémentés dans c sont également remis à zéro dans c (le pire cas pour c étant $\mathtt{ic}^n\mathtt{ric}^n$, qui donne une valeur $2n$). On a obtenu une N-exécution π avec N fini, mais $alt(\pi) = \infty$. $\mathcal{A}$ ne vérifie donc pas la définition des B-QWAA.

Par contraposition, tout B-QWAA vérifie CC.

CC $\implies$ quasi-faible

On suppose que $\mathcal{A}$ satisfait CC. Soit π une n-exécution de $\mathcal{A}$. On va montrer que si $alt(\pi)$ est trop grand, alors $val(\pi) > n$, ce qui serait absurde.

Soit $k := |\Gamma|$.

Soit R donné par le Théorème de Ramsey, tel que si un graphe complet G de taille R a ses arêtes colorées par $2^k - 1$ couleurs, alors il contient une clique de taille $n + 2$. On peut remarquer que R ne dépend que de k et n.

Autrement dit, si k est fixé, il existe une fonction de correction α_R tel que l'on peut toujours prendre $R = \alpha_R(n)$.

Soit $m = alt(\pi)$, et supposons que $m > 2|Q|R$. On peut alors trouver des états $(q_i)_{1 \leq i \leq m}$ tels que π visite q_1 à q_m dans cet ordre, et de plus tous les q_i

avec i pair sont dans F, et tous les q_i avec i impairs sont dans $Q \setminus F$. Soit u le chemin de q_1 à q_m. Pour tout i, on appelle x_i la position de q_i dans u.

D'après la supposition sur m, il existe un ensemble $I \subset [1, m]$ et un état $q \in Q$ tel que $|I| \geq R$, et pour tout $i \in I$, $q_i = q$. Le chemin u contient donc $|I| - 1$ cycles consécutifs de q à q, chacun contenant des états acceptants et refusants. Par la définition de CC, chacun de ces cycles (incluant la concaténation de plusieurs d'entre eux) doit incrémenter l'un des compteurs sans le remettre à zéro.

On va maintenant mettre en application le Théorème de Ramsey, afin d'obtenir $val_B(u) > n$.

Considérons le graphe complet (non orienté) G dont l'ensemble des sommets est $\{x_i : i \in I\}$, de taille au moins R. On définit l'ensemble de couleurs $K = \{A \subseteq \Gamma : A \neq \emptyset\}$. On colorie les arêtes de G de la manière suivante : pour tout $i < j$ dans I, la couleur de l'arête qui relie x_i et x_j est l'ensemble des compteurs qui sont incrémentés mais non remis à zéro entre x_i et x_j. On sait par CC que ces ensembles ne seront jamais vides, ils définissent donc bien des couleurs dans K.

Par choix de R, et puisque $|K| = 2^k - 1$, il existe une clique C de taille $n+2$ dans G, complètement coloriée par un unique $A \in K$. Soit $\gamma \in A$. On peut écrire $C = \{x_{i_1}, \ldots, x_{i_{n+2}}\}$, avec $i_1 < \cdots < i_{n+2}$. Pour tout $j \in [1, n+1]$, γ est incrémenté entre x_{i_j} et $x_{i_{j+1}}$, et il n'est jamais remis à zéro entre x_{i_1} et $x_{i_{n+2}}$. Ceci implique $val(\pi) \geq n+1$, ce qui est absurde.

La supposition $alt(\pi) > 2|Q|R$ débouche sur une contradiction, donc $alt(\pi) \leq 2|Q|R$ pour $R = \alpha_R(n)$, où α_R ne dépend que de k.

En résumé, toute n-exécution π de $\mathcal{A}$ vérifie $alt(\pi) \leq 2|Q|\alpha_R(n)$, ce qui montre que $\mathcal{A}$ est bien un B-QWAA.. □

Finalement, on dira qu'une fonction de coût f est *quasi-faible*, s'il existe un B-QWAA $\mathcal{A}$ tel que $f \approx [\![\mathcal{A}]\!]$.

11.3 Premiers résultats d'expressivité

11.3.1 Etat de l'art sur les fonctions de coûts faibles

L'étude du pouvoir d'expressivité des B-WAA et des S-WAA en terme de fonctions de coût reconnues a été initié par Vanden Boom dans [VB11].

Les résultats suivants y sont montrés :

Théorème 11.3.1 ([VB11]) *Soit f une fonction de coût sur arbres infinis. Alors f est reconnue par un B-WAA si et seulement si f est reconnue par un S-WAA (et la dualisation est effective). De plus, si c'est le cas, alors on peut construire un B-NBA $\mathcal{U}$ et un S-NBA $\mathcal{U}'$ qui reconnaissent tous deux f.*

Ce théorème sont basés sur des résultats de mémoire finie dans les jeux à objectifs B ou S : les stratégies à mémoire finie sont suffisantes pour Eve dans certains B-ABA ou S-ABA. En particulier on dispose du théorème suivant :

Théorème 11.3.2 ([VB11]) *Soit $\mathcal{A}$ un B-WAA. Il existe M tel que limiter Eve aux stratégies à mémoire finie M ne change pas la fonction de coût reconnue par $\mathcal{A}$.*

Vanden Boom définit également la logique de coût faible WCMSO sur les arbres infinis, et montre le théorème suivant :

Théorème 11.3.3 ([VB11]) *Une fonction de coût sur abres infinis est reconnue par un B-WAA si et seulement si elle est définissable dans la logique WCMSO.*

Ce théorème montre une certaine robustesse de la notion de fonctions de coûts faibles sur arbres infinis, car les extension des différents formalismes classiques sont toujours équivalents : dans le cas classique, WMSO et les automates faibles définissent les mêmes langages d'arbres infinis.

On ne s'étendra pas plus ici sur l'aspect logique des fonctions de coût sur abres infinis.

11.3.2 Liens entre fonctions faibles et quasi-faibles

Dans [KV11], Vanden Boom généralise le Théorème 11.3.1 aux B-QWAA :

Théorème 11.3.4 (Simulation) *Soit f une fonction de coût sur arbres infinis reconnues par un B-QWAA. Alors on peut construire un B-NBA $\mathcal{U}$ et un S-NBA $\mathcal{U}'$ qui reconnaissent tous deux f.*

Les mêmes arguments sont toujours utilisables. En effet, la propriété des automates faibles utilisée dans la preuve du Théorème 11.3.1 est la stabilisation des exécutions acceptantes dans des états de Büchi, l'uniformité de la borne sur le nombre d'alternations entre états acceptants et refusants n'est pas utilisée.

En particulier, puisque les inégalités de type $f \preccurlyeq g$ sont toujours décidables si f est donnée par un S-automates et g par un B-automate, on obtient le corollaire suivant :

Corollaire 11.3.5 *Soient f et g des fonctions de coût sur arbres infinis reconnues par des B-QWAA. Alors on peut décider si $f \preccurlyeq g$.*

On a donc étendu la classe des fonctions de coûts sur arbres infinis pour lesquelles la comparaison est décidable.

On montre ce fait de façon plus précise, en prouvant que les B-QWAA sont strictement plus expressifs que les B-WAA et S-WAA :

Théorème 11.3.6 *Il existe une fonction de coût f reconnaissable par un B-QWAA mais par aucun B-WAA.*

Démonstration

On construit explicitement la fonction $f : \mathcal{T}_{\mathbb{A}} \to \mathbb{N}_\infty$ de l'énoncé, sur l'alphabet $\mathbb{A} = \{\vee, \wedge\} \times \{e, a, b\}$. On appelle $\pi_1 : \mathbb{A} \to \{\vee, \wedge\}$ et $\pi_2 : \mathbb{A} \to \{e, a, b\}$ les projections de $\mathbb{A}$ sur ses composantes.

Soit $t \in \mathcal{T}_{\mathbb{A}}$, on dira qu'un ensemble de noeuds $C_t \subseteq \{0, 1\}^*$ est un *arbre de choix* pour t :

- $\epsilon \in C_t$;
- pour tout $x \in C_t$, si $\pi_1(t(x)) = \vee$ alors $x0 \in C_t$ ou $x1 \in C_t$;
- pour tout $x \in C_t$, si $\pi_1(t(x)) = \wedge$ alors $x0 \in C_t$ et $x1 \in C_t$;
- pour tout $x \in C_t$ tel que $\pi_2(t(x)) = a$, tous les chemins de C_t partant de x contiennent une position étiquetée b.

Si C_t est un arbre de choix pour t, on définit sa valeur $val(C_t)$ acomme le nombre maximum de a sur un chemin (pour l'étiquetage défini par t).

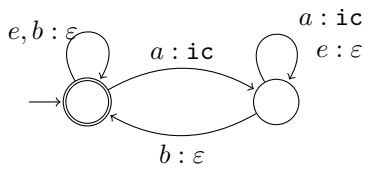
On définit maintenant la fonction de coût f pour tout t par

$$f(t) = \inf \{val(C_t) : C_t \text{ est un arbre de choix pour } t\}.$$

En particulier, s'il n'existe pas d'arbre de choix pour t, alors $f(t) = \infty$.

Il est facile de montrer que f est reconnue par un B-automate quasi-faible, qui est même non-déterministe. On peut en effet définir un automate à 2 états qui devine l'arbre de choix, et compte le nombre de a le long de chacune de ses branches. L'un des états sert à se rappeler que l'on attend un b car un a a été vu, c'est l'état non acceptant.

Autrement dit, on va simuler l'automate suivant sur l'une des deux branches (de manière non-déterministe) si le noeud est étiqueté par $\vee$, et sur les deux branches simultanément s'il est étiqueté par $\vee$:



Cet automate vérifie la caractérisation de la Proposition 11.2.4, donc il est quasi-faible.

On suppose maintenant que f est reconnue par un B-WAA $\mathcal{A}$.Cela signifie qu'il existe une fonction de correction α telle que $f \approx_\alpha \mathcal{A}$.

Les états de $\mathcal{A}$ sont partitionnés en $Q = Q_0 \uplus Q_1 \uplus \cdots \uplus Q_m$, avec $F = \bigcup_{i \text{ pair}} Q_i$, tel chaque cycle appartient à l'un des Q_i, et il n'existe pas de transition $Q_i \to Q_j$ avec $i < j$.

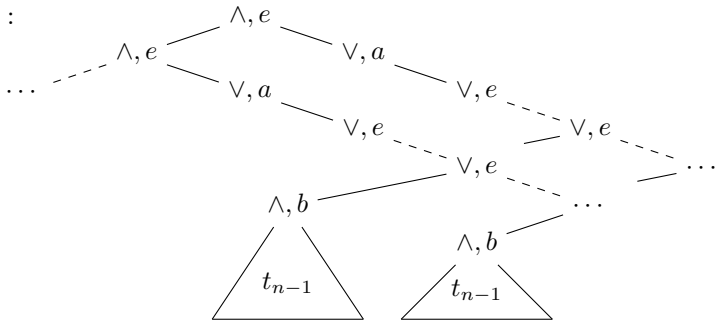
Soit $N = |Q|$.On construit par récurrence une famille $(t_n)_{n \in \mathbb{N}}$ d'arbres de $\mathcal{T}_{\mathbb{A}}$:

L'arbre t_0 est entièrement étiqueté par $(\wedge, e)$. Si t_{n-1} est défini, on construit t_n de la manière suivante :

- $t_n(0^*) = (\wedge, e)$
- $t_n(0^*1) = (\vee, a)$

– $t_n(0^*11^+) = (\vee, e)$
– $t_n(0^*1^N1^+0) = (\wedge, b)$
– les sous-arbres enracinés en $0^*1^N1^+00^+$ sont t_{n-1}
– les autres sous-arbres (pour compléter l'arbre binaire) sont t_0

L'arbre t_n :



On a $f(t_n) = n$ pour tout $n \in \mathbb{N}$.

Le principe qui sous-tend la définitions de ces t_n est de forcer tout B-ABA qui reconnaitraît f à faire $\Theta(n)$ alternations sur t_n.

On va montrer l'existence d'une fonction $\theta : \mathbb{N} \to \mathbb{N}$ avec $\lim_{n\to\infty} \alpha(n) = \infty$, telle que $\mathcal{A}$ doit faire au moins $\theta(n)$ alternations pour accepter t_n.

On définit θ par récurrence. Pour commencer on peut prendre $\theta(0) = 0$.

Soit $n \in \mathbb{N}$, et soit σ une stratégie d'Eve dans $(\mathcal{A}, t_n)$ de valeur au plus $\alpha(n)$. Toute exécution compatible avec σ doit stabiliser dans une partition acceptante. La stratégie σ fixe un arbre de choix C_t pour t_n, et chaque branche de C_t correspond à une $\alpha(n)$-exécution acceptante $\mathcal{A}$. En particulier, sur la branche 0^ω, $\mathcal{A}$ doit stabiliser dans un Q_i acceptant (i pair), à partir d'un noeud de la forme 0^d. Puisque ce noeud est étiqueté $\wedge$, le noeud 0^d1 doit être dans C_t.

Eve est ensuite forcée d'atteindre le noeud 0^d1^{N+1} pour que C_t soit un arbre de choix : il faut un b après le a vu en 0^d1. Par définition de N, il y a un cycle c entre 0^d1 et 0^d1^{N+1}, aux extrémités duquel l'état q de $\mathcal{A}$ est les même.

Puisque $\mathcal{A}$ est un automate faible, pour tout compteur γ, si c comporte un $\texttt{ic}_\gamma$ alors il comporte un $\texttt{r}_\gamma$. De plus, tout le cycle est inclus dans une partition Q_j, avec $j \leq i$.

Si Q_j est acceptante, alors on considère l'arbre t' obtenu en répétant ce cycle une infinité de fois. Sur la branche ainsi obtenue, le a présent au noeud 0^d1 n'est suivi par aucun b, donc $f(t') = \infty$. Cependant, Eve possède une stratégie dans $(\mathcal{A}, t')$ lui permettant d'accepter t' avec une valeur au plus $2\alpha(n)$, en répétant les même choix indéfiniment sur le cycle c.

C'est absurde, donc Q_j est forcément refusante (j est impair).

Eve doit donc choisir l'un des b situés sur les noeuds $0^d1^N1^+0$, sinon elle reste sur la branche 0^d1^ω, et on peut de nouveau obtenir une contradiction).

Il reste à accepter t_{n-1} depuis la partition Q_j, ce qui nécessite $\theta(n-1)$ alternations par hypothèses de récurrence. On a donc bien besoin de $\theta(n) := \theta(n-1) + 1$ alternations dans $\mathcal{A}$ pour accepter l'arbre t_n.

La fonction construire est $\theta(n) = n$, et tout B-WAA qui reconnaît f a besoin d'au moins n partitions, pour tout $n \in \mathbb{N}$ (il doit se comporter de manière cohérente sur tous les t_n). On peut en conclure qu'un tel automate n'existe pas. □

Chapitre 12

BS-automates

Avant de pouvoir démontrer le résultat principal de cette partie, on a besoin d'introduire des objets et de démontrer des lemmes qui seront utilisés plus tard.

Dans cette partie, on va définir et étudier les BS-automates alternants sur arbres infinis (BS-NBA), ainsi que des transducteurs agissant sur des mots infinis de BS-actions (BS-T). Ce sont des automates qui possèdent simultanément les deux types de compteurs : B et S. Ces automates serviront dans la prochaine section à étudier l'automate produit obtenu à partir d'un B-NBA et d'un S-NBA.

12.1 BS-NBA sur arbres infinis

12.1.1 Compteurs

On veut définir des automates possédant les deux types de compteurs, sur mots infinis et arbres infinis. L'ensemble des compteurs est séparé en $\Gamma = \Gamma_B \uplus \Gamma_S$, pour différencier le type des compteurs.

On écrira $\mathbb{C}$ pour l'alphabet des actions de compteurs : $\mathbb{C} := \mathbb{C}_B^{\Gamma_B} \times \mathbb{C}_S^{\Gamma_S}$.

Les fonctions val_B et val_S définies auparavant sur $(\mathbb{C}_B^{\Gamma_B})^\omega$ et $(\mathbb{C}_S^{\Gamma_S})^\omega$ sont étendues à $\mathbb{C}^\omega$, en projetant simplement sur la composante concernée (on ignore les compteurs de l'autre type).

Compteurs hiérarchiques

Si $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$, on définit un un sous-ensemble $\mathbb{C}_h \subseteq \mathbb{C}$: les actions *hiérarchiques*. Une action est hiérarchique si elle respecte la contrainte suivante sur les compteurs : lorsqu'un compteur i est manipulé (par une action différente de ε), tous les compteurs de numéro plus grand que lui doivent être remis à zéro.

Ainsi, ν est une action de CH si pour tout $i \in [1, k]$ tel que $\nu(\gamma_i) \neq \varepsilon$, alors pour tout $j > i$, $\nu(\gamma_j) = \mathtt{r}$. Cela signifie que si $\nu \in \mathbb{C}_h$, il y a au plus un compteur sur lequel ν n'effectue ni ε, ni $\mathtt{r}$. Il suffit de décrire l'action de ν sur ce compteur pour la spécifier complètement.

Notons que cette contrainte mélange les compteurs des deux types, et Γ est totalement ordonné : une action sur un B-compteur peut forcer une remise à zéro d'un S-compteurs.

Par exemple si $k = 4$, on notera $\mathtt{ic}_2$ pour l'action $(\varepsilon, \mathtt{ic}, \mathtt{r}, \mathtt{r})$, et $\mathtt{cr}_3$ pour l'action $(\varepsilon, \varepsilon, \mathtt{cr}, \mathtt{r})$, en supposant que γ_2 est un B-compteur et γ_3 est un S-compteur.

12.1.2 BS-automates alternants sur arbres infinis

Un BS-NBA est un tuple $\mathcal{A} = \langle Q, \mathbb{A}, In, F_B, F_S, \Gamma_B, \Gamma_S, \Delta \rangle$.

Il y a maintenant deux ensembles d'états de Büchi : F_B et F_S. En effet, on considèrera maintenant que l'automate définit plusieurs sémantiques, relatives aux différents types de compteurs.

L'ensemble de transitions Δ est un sous-ensemble de $Q \times \mathbb{A} \times (\mathbb{C} \times Q)^2$, comme dans le cas des B-NBA et S-NBA sur abres infinis.

Une exécution de $\mathcal{A}$ sur un arbre $t \in \mathcal{T}_{\mathbb{A}}$ est définie comme précédemment : c'est un étiquetage de t par $\mathbb{C} \times Q$, compatible avec les lettres de π et la fonction de transition δ.

On dira qu'une telle exécution est B-acceptante (resp. S-acceptante), si toutes ses branches sont acceptantes, avec F_B (resp. F_S) comme états de Büchi.

Si $\Gamma = \Gamma_B \uplus \Gamma_S = \{\gamma_1, \ldots, \gamma_k\}$ est totalement ordonné, et que toutes les actions de $\mathcal{A}$ sont dans $\mathbb{C}_h$ (défini dans la section précédente), on dit que $\mathcal{A}$ est un BS-automate *hiérarchique*, noté hBS-NBA.

12.1.3 Sémantiques des BS-NBA

Soit $\mathcal{A} = \langle Q, \mathbb{A}, In, F_B, F_S, \Gamma_B, \Gamma_S, \Delta \rangle$ un BS-NBA sur arbres infinis.

L'objectif à atteindre pour l'automate est plus difficile à définir ici : il faut optimiser deux valeurs différentes (B et S) au lieu d'une seule comme dans tous les modèles précédents.

Une exécution de $\mathcal{A}$ est un étiquetage de l'arbre d'entrée t par $\mathbb{C} \times Q$, compatible avec les lettres de t et la fonction de transition Δ (la racine étant étiquetée par un état initial).

On dira qu'une telle exécution est B-acceptante (resp. S-acceptante), si elle contient une inifinité d'états de F_B (resp. F_S) sur toutes les branches. La B-valeur val_B et S-valeur val_S d'une exécution sont définies comme précédemment.

En ignorant Γ_S et F_S, et en considérant $\mathcal{A}$ comme un B-NBA, on peut définir $[\![\mathcal{A}]\!]_B$: on tente de minimiser la B-valeur de l'exécution, tout en garantissant qu'elle est B-acceptante.

$$[\![\mathcal{A}]\!]_B(t) = \sup val_B(\sigma) : \sigma \text{ stratégie pour Eve dans } (\mathcal{A}, t)$$

Cependant, on veut s'intéresser aussi aux S-valeurs, mais pas de manière indépendante, pour rendre compte des exécutions qui offrent un bon compromis entre les deux sémantiques.

L'idée est de considérer la S-sémantique de l'automate $\mathcal{A}$, mais lorsque celui-ci est restreint aux exécutions dont la B-valeur n'excède pas une certaine borne m.

Ceci fixe donc une S-sémantique pour chaque borne m sur la B-sémantique, au lieu d'une unique S-sémantique pour tout l'automate.

On définit donc une nouvelle sémantique $[\![\mathcal{A}]\!]^B_S$, qui est maintenant une fonction $\mathbb{N} \to \mathbb{A}^\omega \to \mathbb{N}_\infty$: on doit spécifier la borne m pour obtenir la fonction $\mathbb{A}^\omega \to \mathbb{N}_\infty$ attendue.

Pour tout $m \in \mathbb{N}$ et $u \in \mathbb{C}^\omega$, on définit

$val^m_S(u) = \begin{cases} val_S(u) \text{ si } val_B(u) \leq m \\ 0 \text{ sinon} \end{cases}$

Cela reflète le fait que si l'on dépasse le seuil m fixé, l'exécution est un échec, ce qui correspond à une S-valeur nulle.

On pose finalement

$$[\![\mathcal{A}]\!]^B_S(m)(t) := \sup \{val^m_S(R) : R\text{exécution de } \mathcal{A} \text{ sur } t\}.$$

Remarque 12.1.1 *Si $m \leq m'$, alors $[\![\mathcal{A}]\!]^B_S(m) \leq [\![\mathcal{A}]\!]^B_S(m')$. En effet, en augmentant la borne m, on remplace la valeur 0 associée à certaines exécutions par leur véritable S-valeur, ce qui peut faire augmenter la S-sémantique.*

12.1.4 Equivalence de BS-automates

On veut maintenant donner une notion d'équivalence entre deux BS-NBA $\mathcal{A}$ et $\mathcal{A}'$.

Si α est une fonction de correction, on dira que $\mathcal{A}$ et $\mathcal{A}'$ sont α-équivalents, noté $\mathcal{A} \approxeq_\alpha \mathcal{A}'$,si

- $[\![\mathcal{A}]\!]_B \approx_\alpha [\![\mathcal{A}']\!]_B$,
- pour tout $m \in \mathbb{N}$, il existe β_m tel que
  - $[\![\mathcal{A}]\!]^B_S(m) \preccurlyeq_{\beta_m} [\![\mathcal{A}']\!]^B_S(\alpha(m))$, et
  - $[\![\mathcal{A}']\!]^B_S(m) \preccurlyeq_{\beta_m} [\![\mathcal{A}]\!]^B_S(\alpha(m))$.

On dira que $\mathcal{A}$ et $\mathcal{A}'$ sont *BS-équivalents*, noté $\mathcal{A} \approxeq \mathcal{A}'$, s'il existe α tel que $\mathcal{A} \approxeq_\alpha \mathcal{A}'$.

L'idée est que l'on veut préserver la B-sémantique entre $\mathcal{A}$ et $\mathcal{A}'$, et l'on veut en plus garantir qu'une fois une borne fixée pour les B-compteurs, la S-sémantique est aussi préservée.

Toutes ces définitions sont identiques pour les BS-automates sur abres infinis. Leur sémantique $[\![\mathcal{A}]\!]^B_S$ est une fonction $\mathbb{N} \to \mathcal{T}_\mathbb{A} \to \mathbb{N}_\infty$.

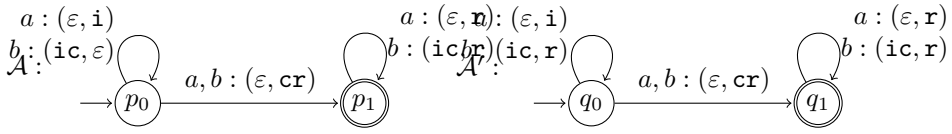
12.1.5 Exemple

Description de deux BS-NBA $\mathcal{A}$ et $\mathcal{A}'$

Soit $\mathbb{A} := \{a, b, c\}$. On décrit dans cette section deux BS-NBA $\mathcal{A}$ et $\mathcal{A}'$ sur $\mathcal{T}_\mathbb{A}$. Ces deux automates ont chacun un B-compteur et un S-compteur. De plus leurs états B-acceptants F_B et S-acceptants F_S sont confondus, et sont représentés par des doubles cercles.

Pour simplifier l'exemple, on considèrera ici que les automates ne lisent que la branche gauche de l'arbre t donnée en entrée.

On écrit $a : (d, d')$ pour décrire une transition étiquetée a, avec action d (resp. d') sur le B- (resp. S-) compteur. Des boucles $c : (\varepsilon, \varepsilon)$ sont présentes sur tous les états, on omet ces boucles sur le schéma.



La seule différence entre $\mathcal{A}$ et $\mathcal{A}'$ est l'action de b sur l'état initial : $(\mathtt{ic}, \varepsilon)$ pour $\mathcal{A}$ et $(\mathtt{ic}, \mathtt{r})$ pour $\mathcal{A}'$.

On peut remarquer que les actions de $\mathcal{A}'$ sont hiérarchiques : elles sont toutes de la forme (ε, ν) ou $(\nu, \mathtt{r})$. $\mathcal{A}'$ est donc un hBS-NBA.

Sémantiques de $\mathcal{A}$ et $\mathcal{A}'$

On peut d'abord remarquer qu'en ignorant les S-compteurs, on obtient $[\![\mathcal{A}]\!]_B = [\![\mathcal{A}']\!]_B = |\cdot|_b$: le B-compteur compte simplement le nombre de b dans les deux automates, et ce peu importe les choix de transitions effectués par l'automate.

De plus, en ignorant le B-compteur de $\mathcal{A}$, on obtient $[\![\mathcal{A}]\!]_S = |\cdot|_a$. En effet, si le nombre de a est fini, la meilleure exécution pour $\mathcal{A}$ est d'attendre le dernier a pour passer dans l'état p_1, enregistrant ainsi le nombre de a vus dans le mot. Si le nombre de a est infini, pour chaque n il existe une exécution S-acceptante de valeur n. La S-sémantique étant définie comme le supremum de toutes ces exécutions, $[\![\mathcal{A}]\!]_S(u) = \infty$ si $|u|_a = \infty$.

Cependant, dans $\mathcal{A}'$, chaque b lu dans q_0 provoque une remise à zéro du S-compteur qui compte le nombre de a. Il n'y a a priori pas de borne sur le nombre de b en tout, donc $[\![\mathcal{A}']\!]_S \not\approx [\![\mathcal{A}]\!]_S$. Mais si l'on fixe le nombre de b, et donc la B-valeur de $\mathcal{A}$ et $\mathcal{A}'$, alors on peut mettre en relation $[\![\mathcal{A}]\!]_S$ et $[\![\mathcal{A}]\!]_S$. Plus précisément, si $u \in \mathbb{A}^\omega$ est tel que $[\![\mathcal{A}]\!]_B(u) \leq m$, alors $[\![\mathcal{A}]\!]_S(u) \approx_{\beta_m} [\![\mathcal{A}']\!]_S(u)$, avec $\beta_m(n) = (m+1)n$. Ce fait est facile à constater : tout d'abord on a toujours $[\![\mathcal{A}]\!]_S(u) \geq [\![\mathcal{A}']\!]_S(u)$. Ensuite, si l'on remet à zéro au plus m fois le S-compteur dans $\mathcal{A}'$, alors $[\![\mathcal{A}]\!]_S(u) \leq (m+1) * [\![cA']\!](u)$. Ceci montre que pour tout m, $[\![\mathcal{A}]\!]^B_S(m) \approx_{\beta_m} [\![\mathcal{A}']\!]^B_S(m)$, ce qui est un cas particulier de la relation de BS-équivalence, avec $\alpha = id$.

On a donc montré $\mathcal{A} \cong_{id} \mathcal{A}'$.

12.2 Transducteurs de BS-actions sur mots infinis

12.2.1 Définition

On définit dans cette section un autre type de BS-automates : les BS-transducteurs non-déterministes sur mots infinis d'actions, notés BS-T.

On se fixe un alphabet d'entrées de BS-actions : $\mathbb{C} = \mathbb{C}_B^{K_B} \times \mathbb{C}_S^{K_S}$, avec $K_B, K_S \in \mathbb{N}$.

L'objectif d'un BS-T avec compteurs Γ_B, Γ_S est de lire un mot de $\mathbb{C}^\omega$, et de renvoyer un mot de $(\mathbb{C}')^\omega$, où $\mathbb{C}' = \mathbb{C}_B^{|\Gamma_B|} \times \mathbb{C}_S^{|\Gamma_S|}$.

Un BS-T est donc tuple $\mathcal{A} = \langle Q, \mathbb{A}, q_0, \Gamma_B, \Gamma_S, \Delta \rangle$.

Puisqu'on considère seulement les BS-T non-déterministes, on peut exprimer la fonction de transition sous la forme d'un ensemble $\Delta \subseteq Q \times \mathbb{A} \times \mathbb{C} \times Q$. Remarquons que $\mathcal{A}$ ne possède pas de conditions d'acceptation. On les évite volontairement car on ne s'en servira pas ici, mais on pourrait ajouter des états de Büchi F_B et F_S comme précédemment. Remarquons cependant que Δ n'est pas supposé complet : l'automate peut se retrouver bloqué dans un état q en lisant une lettre a, si $\Delta \cap \{q\} \times \{a\} \times \mathbb{C} \times Q = \emptyset$. Ceci correspond à un échec de l'exécution. On considère donc seulement les exécutions de longueur ω dans la suite.

On dira que $\mathcal{A}$ est hiérarchique si toutes ses transitions sont étiquetés par des actions hiérarchiques de $\mathbb{C}'$. On notera hBS-T pour désigner un BS-T hiérarchique.

Une exécution ρ de $\mathcal{A}$ sur un mot $u \in \mathbb{C}^\omega$ est une séquence de Δ^ω compatible avec u, utilisant q_0 comme état initial, et respectant la correspondance entre les états de départ et d'arrivée de chaque transition.

On écrira $val_B(\rho)$ (resp. $val_S(\rho)$) la B-valeur (resp. S-valeur) du mot de $\mathbb{C}'^\omega$ induit par ρ.

Si $m \in \mathbb{N}$, on dira que ρ est une m-exécution si $val_B(\rho) \leq m$

12.2.2 Fidélité

Soit $\mathcal{A}$ un BS-T de $\mathbb{C}$ vers $\mathbb{C}'$.

Comme précédemment, on veut définir $[\![\mathcal{A}]\!]_S^B : \mathbb{N} \to \mathbb{A}^\omega \to \mathbb{N}_\infty$, en imposant une borne sur la B-valeur. Cependant, dans ce cas, le mot lu en entrée possède également une B-valeur, et il est plus simple de considérer celle-ci, pour éviter d'avoir à prendre en compte l'approximation effectuée par $\mathcal{A}$ entre les B-valeurs d'entrée et de sortie.

Pour tout $m \in \mathbb{N}$, on définit $L_m = \{u \in \mathbb{C}^\omega : val_B(u) \leq m\}$.

On peut maintenant définir $[\![\mathcal{A}]\!]_S^B(m)$ comme la restriction de $[\![\mathcal{A}]\!]_S$ à L_m : pour tout $u \in \mathbb{C}^\omega$, on pose

$$[\![\mathcal{A}]\!]_S^B(m)(u) = \begin{cases} [\![\mathcal{A}]\!]_S(u) \text{ si } val_B(u) \leq m \\ \infty \text{ sinon} \end{cases}$$

Pour tout $m \in \mathbb{N}$ et $u \in \mathbb{C}^\omega$, on définit également

$val_S^m(u) = \begin{cases} val_S(u) \text{ si } val_B(u) \leq m \\ \infty \text{ sinon} \end{cases}$

Soit α une fonction de correction. On dit que $\mathcal{A}$ est *α-fidèle* si pour tout mot $u \in \mathbb{C}^\omega$,

- Toute exécution ρ de $\mathcal{A}$ sur u vérifie $val_B(\rho) \approx_\alpha val_B(u)$
- Pour tout $m \in \mathbb{N}$, il existe β_m tel que $[\![\mathcal{A}]\!]_S^B(m) \approx_{\beta_m} val_S^m$

On dira que $\mathcal{A}$ est *fidèle* s'il existe α tel que $\mathcal{A}$ est α-fidèle.

Notons encore une fois que contrairement à la section précédente, cette sémantique utilise la B-valeur du mot d'entrée comme borne. Ceci évite de prendre en considération l'α-approximation introduite par le transducteur, et simplifie la définition de la sémantique.

La notion de fidélité est plus forte que la notion de BS-équivalence. En effet, Eve ne jouant aucun rôle, on empêche Adam d'influer sur la B-valeur de l'exécution en imposant le fait que *toutes* les exécutions de l'automate aient des B-valeurs équivalentes. Ainsi le seul objectif d'Adam est de maximiser la S-valeur, sans prêter attention à la B-valeur, puisque celle-ci ne dépend pas de ses choix.

12.3 Déterminisme en histoire

On va maintenant s'intéresser à une propriété appelée déterminisme en histoire. Cette notion a été introduite par Colcombet dans [Col09] pour les B-automates et les S-automates, et on la généralise ici aux BS-T.

Cette notion est un affaiblissement de la notion de déterminisme, qui permet de conserver quelques propriétés désirables, comme la possibilité de composer un automate avec un jeu. Cette propriété est donc désirable pour un transducteur fidèle que l'on souhaite composer avec un BS-NBA, tout en conservant sa sémantique.

On commence par définir cette notion sur les automates de coûts possédant un seul type de compteurs, telle qu'elle est introduite dans [Col09].

12.3.1 B- et S-automates déterministes en histoire

Rappelons que les automates de coûts, même sur mots finis, ne peuvent pas toujours être déterminisés. En revanche, la notion plus faible de « déterminisme en histoire » (hd) permet d'obtenir certaines propriétés de composition des automates de coût.

Pour choisir la transition à emprunter, un automate déterministe n'a besoin de connaitre que l'état courant q, et la lettre $a \in \mathbb{A}$ lue en entrée. Les informations nécessaires pour déterminer la prochaine transition d'un automate déterministe en histoire sont :

- la configuration (q, a) comme pour l'automate déterministe,
- une valeur n fixée, que l'on utilise comme une borne pour l'exécution,
- l'historique de l'exécution.

En pratique, l'historique de l'exécution sera utilisé seulement pour connaître la valeur courante des compteurs.

On donne maintenant une définition plus formelle. Soit $\mathcal{A}$ un B-automate ou S-automate non-déterministe, avec table de transitions $\Delta \subseteq Q \times \mathbb{A} \times \mathbb{C} \times Q$, et un seul état initial q_0

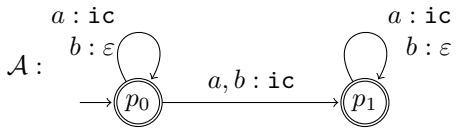
Une *stratégie de lecture* pour $\mathcal{A}$ est une fonction $\delta : \mathbb{A}^* \times Q \times \mathbb{A} \to \mathbb{C} \times Q$ telle que pour tout $u \in \mathbb{A}^*$ et $(q, a) \in Q \times \mathbb{A}$, $\delta(u, q, a) \in \Delta(q, a)$.

Une exécution R de $\mathcal{A}$ est *dirigée par* δ si, étant dans la configuration (q, a) après avoir lu un mot u, la prochaine transition de R est $\delta(u, q, a)$. Puisqu'il n'y a qu'un seul état initial dans $\mathcal{A}$, la donnée de u et δ fixe une seule exécution de $\mathcal{A}$ sur u dirigée par δ.

Si $\mathcal{A}$ est un B-automate (resp. S-automate), on dit que $\mathcal{A}$ est *déterministe en histoire* (hd) s'il existe une famille de stratégies de lecture $\delta = (\delta_n)_{n \in \mathbb{N}_\infty}$ telle que pour tous $n \in \mathbb{N}$ et $u \in \mathbb{A}^\omega$, si R_n est l'exécution de $\mathcal{A}$ sur u dirigée par δ_n, alors $val_B(R_n) \leq n$ (resp. $val_S(R_n) > n$). De plus, si $\mathcal{A}^\delta$ est l'automate $\mathcal{A}$ restreint aux exécution R_n, on a $\mathcal{A}^\delta \approx \mathcal{A}$.

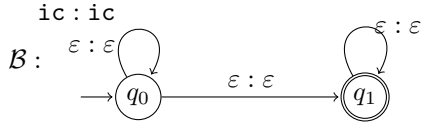
La principale raison de s'intéresser aux automates hd est leur capacité à être composé avec des jeux, et donc des automates alternants. On peut en particulier considérer un automate $\mathcal{B}$ lisant ce qui est produit par un autre automate alternant $\mathcal{A}$ (actions de compteurs, états de Büchi,...), et produisant ses propres actions en réponse. Si un tel automate $\mathcal{A}$ est hd, la composition des deux se passera de la manière attendue, et on pourra utiliser $\mathcal{B}$ pour évaluer les actions de $\mathcal{A}$. Cette construction ne fonctionne pas si $\mathcal{B}$ est non-déterministe, car $\mathcal{B}$ pourrait avoir besoin de connaître l'avenir du mot pour produire les actions demandées. Si Adam joue un rôle dans $\mathcal{A}$, alors il peut modifier les actions produites pour fausser les décisions prises par $\mathcal{B}$ dans la manière de les lire.

Exemple 12.3.1 *Soit $\mathcal{A}$ le B-ABA suivant sur $\mathbb{A} = \{a, b\}$, entièrement contrôlé par Adam :*



Cet automate compte les a, et permet à Adam de faire un `ic` *supplémentaire à n'importe quel moment du mot. La fonction calculée est donc $[\![\mathcal{A}]\!]_B(u) = |u|_a + 1$.*

On veut maintenant analyser le mot d'actions produit par cet automate, au moyen de l'automate non-déterministe $\mathcal{B}$ suivant , sur l'alphabet $\{\mathtt{ic}, \varepsilon\}$:



Cet automate doit deviner un moment où le dernier incrément est passé, pour passer dans l'état acceptant q_1. Si la B-valeur d'un mot $u \in \{\mathtt{ic}, \varepsilon\}^\omega$ est finie, un tel moment existe toujours, et on a donc $[\![\mathcal{B}]\!]_B = val_B$.

Cependant, si l'on regarde le B-ABA $\mathcal{C}$ obtenu en composant $\mathcal{A}$ et $\mathcal{B}$, on s'aperçoit que sa sémantique n'est pas la composition de celles de $\mathcal{A}$ et $\mathcal{B}$.

En effet, Adam a accès à l'état courant dans l'automate $\mathcal{C}$, et peut donc atteindre que $\mathcal{B}$ soit passé dans q_1 pour ajouter un incrément artificiel, induisant

ainsi un échec de l'exécution de $\mathcal{B}$. On a donc $[\![\mathcal{C}]\!] = \infty$, alors qu'on voudrait $[\![\mathcal{C}]\!] \approx [\![\mathcal{A}]\!]$, puisque $[\![\mathcal{B}]\!] = val_B$.

Ceci provient du fait que $\mathcal{B}$ n'est clairement pas hd : il doit deviner une information sur le futur pour décider quelle transition prendre.

12.3.2 BS-T déterministes en histoire

On définira ici la notion de déterminisme en histoire seulement pour les BS-T fidèles. Cette notion n'a pas forcément de sens sur les BS-T en général, car leur sémantique n'est pas clairement définie.

On dira qu'un BS-T fidèle $\mathcal{A}$ de $\mathbb{C}$ vers $\mathbb{C}'$ est *déterministe en histoire* (hd) si

- il existe une famille de stratégies $\delta = (\delta_n)_{n\in\mathbb{N}_\infty}$ telle que pour tous $n \in \mathbb{N}$ et $u \in \mathbb{C}^\omega$, si R_n est l'exécution de $\mathcal{A}$ sur u dirigée par δ_n, alors $val_S(R_n) = n$ ou R_n est finie (auquel cas l'exécution de $\mathcal{A}$ se bloque, et $val_S(R_n) = 0$).
- Si $\mathcal{A}^\delta$ est l'automate $\mathcal{A}$ restreint aux exécutions R_n, on a pour tout $m \in \mathbb{N}$, il existe β_m tel que $[\![\mathcal{A}]\!]^B_S(m) \preccurlyeq_{\beta_m} [\![\mathcal{A}^\delta]\!]^B_S(m)$.
- la famille δ est *monotone*, c'est-à-dire que pour tout u, si R_n est infinie, alors pour tout $k < n$, R_k est aussi infinie. De plus, R_0 est toujours infinie.

La deuxième condition ne nécessite qu'une inégalité, car l'inégalité inverse est toujours vérifiée. En effet, $\mathcal{A}^\delta$ dispose de moins d'exécutions que $\mathcal{A}$, donc pour tout $m \in \mathbb{N}$ on a $[\![\mathcal{A}^\delta]\!]^B_S(m) \leq [\![\mathcal{A}]\!]^B_S(m)$.

Lemme 12.3.2 *Si $\mathcal{A}$ est un BS-T hd, en reprenant les notations de la définition, on a pour tout $m, k \in \mathbb{N}$ et $u \in L_m$ (c'est-à-dire $val_B(u) \leq m$), si $\beta_m(k) \leq [\![\mathcal{A}]\!]_S(u)$, alors R_k est infinie (et donc acceptante).*

Démonstration Soit $n = [\![\mathcal{A}]\!]^B_S(m)$. On sait que $[\![\mathcal{A}]\!]^B_S(m) \preccurlyeq_{\beta_m} [\![\mathcal{A}^\delta]\!]^B_S(m)$, donc il existe une exécution R_k de $\mathcal{A}^\delta$ sur u telle que $\beta_m(val_S(R_k)) \geq [\![\mathcal{A}]\!]_S(u)$.

Or, $val_S(R_k) = k$ donc il existe k telle que $\beta_m(k) \geq [\![\mathcal{A}]\!]_S(u)$ et R_k acceptante. Par monotonie de δ, tout k tel que $\beta_m(k) \leq [\![\mathcal{A}]\!]_S(u)$ vérifie R_k acceptante. □

12.3.3 Composition d'un BS-NBA et d'un BS-T

Soit $\mathcal{A} = \langle Q_\mathcal{A}, \mathbb{A}, In_\mathcal{A}, F_B, F_S, \Gamma_B, \Gamma_S, \Delta_\mathcal{A}\rangle$ un BS-NBA sur $\mathcal{T}_\mathbb{A}$. Soit $\mathbb{C} = \mathbb{C}_B^{|\Gamma_B|} \times \mathbb{C}_S^{|\Gamma_S|}$ l'alphabet des actions effectuées par $\mathcal{A}$.

On a donc $\Delta_\mathcal{A} \subseteq Q_\mathcal{A} \times \mathbb{A} \times (\mathbb{C} \times Q_\mathcal{A})^2$

Soit $\mathcal{H} = \langle Q_\mathcal{H}, \mathbb{C}, q_0, \Gamma'_B, \Gamma'_S, \Delta_\mathcal{H}\rangle$ un BS-T sur $\mathbb{C}$ avec actions dans $\mathbb{C}' = \mathbb{C}_B^{|\Gamma'_B|} \times \mathbb{C}_S^{|\Gamma'_S|}$. On a $\Delta_\mathcal{H} \subseteq Q_\mathcal{H} \times \mathbb{C} \times \mathbb{C}' \times Q_\mathcal{H}$.

On définit $\mathcal{A} \circ \mathcal{H} = \langle Q', \mathbb{A}, In', F'_B, F'_S, \Gamma'_B, \Gamma'_S, \Delta'\rangle$ en posant :

- $Q' = Q_\mathcal{A} \times Q_\mathcal{H}$,
- $In' = In_\mathcal{A} \times \{q_0\}$,
- $F'_B = F_B \times Q_\mathcal{H}$, $F'_S = F_S \times Q_\mathcal{H}$,
- $\Delta' = \{((p,q), a, (\nu_1, (p_1, q_1)), (\nu_2, (p_2, q_2))) :$ il existe $c_1, c_2 \in \mathbb{C}$ tels que $(p, a, (c_1, p_1), (c_2, p_2)) \in \Delta_\mathcal{A}, (p, a, c_1, q_1) \in \Delta_\mathcal{H}, (p, a, c_2, q_2) \in \Delta_\mathcal{H}\}$.

Le Lemme suivant permet de composer un BS-T fidèle déterministe en histoire avec un BS-NBA, en préservant la sémantique de ce dernier.

Théorème 12.3.3 *Soit $\mathcal{A}$ un BS-NBA sut $\mathcal{T}_{\mathbb{A}}$ produisant des actions sur $\mathbb{C}$, et $\mathcal{H}$ un BS-T sur $\mathbb{C}$, α-fidèle et déterministe en histoire. Alors l'automate $\mathcal{C} = \mathcal{H} \circ \mathcal{A}$ obtenu en composant ces deux automates est un BS-NBA tel que $\mathcal{C} \approxeq_\alpha \mathcal{A}$. De plus, si $\mathcal{H}^\delta$ est le BS-T $\mathcal{H}$ restreint aux stratégies hd, alors $\mathcal{C} \approxeq \mathcal{H}^\delta \circ \mathcal{A} \approxeq \mathcal{A}$.*

Démonstration On reprend tous les objets définis par l'énoncé, et on cherche à montrer $\mathcal{C} \approxeq_\alpha \mathcal{A}$.

Il faut d'abord montrer $[\![\mathcal{C}]\!]_B \approx_\alpha [\![\mathcal{A}]\!]_B$.

Soit $t \in \mathcal{T}_{\mathbb{A}}$ un arbre infini. Soit $m = [\![\mathcal{C}]\!]_B(t)$, témoigné par une exécution $R_\mathcal{C}$ de $\mathcal{C}$ sur t telle que $val_B(R_\mathcal{C}) \leq m$.

Par définition de $\mathcal{C} = \mathcal{H} \circ \mathcal{A}$, $R_\mathcal{C}$ induit en particulier une exécution $R_\mathcal{A}$ de $\mathcal{A}$ sur t. On montre que cette exécution convient. Soit π une branche de $R_\mathcal{A}$, et τ la branche correspondante de $R_\mathcal{C}$. τ correspond donc à une exécution de $\mathcal{H}$ sur π, et par définition de $R_\mathcal{C}$, $val_B(\tau) \leq m$ et $val_S(\tau) \geq n$. Par fidélité de $\mathcal{H}$, on obtient $val_B(\tau) \approx_\alpha val_B(\pi)$, et donc en particulier $val_B(\pi) \leq \alpha(m)$. C'est vrai pour toute branche π, donc $val_B(R_\mathcal{A}) \leq \alpha(m)$, ce qui entraîne $[\![\mathcal{A}]\!]_B(t) \leq \alpha(m)$. On a montré $[\![\mathcal{A}]\!]_B \preccurlyeq_\alpha [\![\mathcal{C}]\!]_B$.

Réciproquement, soit $t \in \mathcal{T}_{\mathbb{A}}$ un arbre infini, et $m = [\![\mathcal{A}]\!]_B(t)$, témoigné par une exécution $R_\mathcal{A}$ de $\mathcal{A}$ sur t telle que $val_B(R_\mathcal{A}) \leq m$. On définit une exécution $R_\mathcal{C}$ de $\mathcal{C}$ sur t en combinant $R_\mathcal{A}$ avec la stratégie δ_0 pour l'automate $\mathcal{H}$. Par définition du déterminisme en histoire, $R_\mathcal{C}$ est bien définie et toutes ses branches sont infinies. Par fidélité de $\mathcal{H}$, $val_B(R_\mathcal{C}) \leq \alpha(m)$. Il s'ensuit $[\![\mathcal{C}]\!]_B \preccurlyeq_\alpha [\![\mathcal{A}]\!]_B$.

On peut donc conclure $[\![\mathcal{C}]\!]_B \approx_\alpha [\![\mathcal{A}]\!]_B$.

Il reste à montrer la deuxième partie de la BS-équivalence : pour tout $m \in \mathbb{N}$, il existe β_m tel que

- $[\![\mathcal{C}]\!]^B_S(m) \preccurlyeq_{\beta_m} [\![\mathcal{A}]\!]^B_S(\alpha(m))$ (1)
- $[\![\mathcal{A}]\!]^B_S(m) \preccurlyeq_{\beta_m} [\![\mathcal{C}]\!]^B_S(\alpha(m))$ (2)

Soit δ la stratégie déterministe en histoire de $\mathcal{H}$. On considère les fonctions de corrections β^{hd}_m données par la définition du déterminisme en histoire de $\mathcal{H}$. On utilisera également les fonctions β^f_m données par la fidélité de $\mathcal{H}$.

On commence par montrer la partie (1). Soit $t \in \mathcal{T}_{\mathbb{A}}$ un arbre infini, et $m \in \mathbb{N}$. Soit $n = [\![\mathcal{C}]\!]^B_S(m)(t)$. On veut montrer qu'il existe une exécution de $\mathcal{A}$ sur t témoignant $[\![\mathcal{A}]\!]^B_S(\alpha(m)) \geq_{\beta^1_m} n$, pour un certain β^1_m.

Soit $R_\mathcal{C}$ une exécution de $\mathcal{C}$ sur t témoignant $n = [\![\mathcal{C}]\!]^B_S(m)(t)$. Cela signifie que $val_B(R_\mathcal{C}) \leq m$ et $val_S(R_\mathcal{C}) \geq n$.

De même que précédemment, $R_\mathcal{C}$ induit une exécution $R_\mathcal{A}$ de $\mathcal{A}$ sur t, telle que $val_B(R_\mathcal{A}) \leq \alpha(m)$, par fidélité de $\mathcal{H}$. On montre que cette exécution convient.

Soit π une branche de $R_\mathcal{A}$, et τ la branche correspondante de $R_\mathcal{C}$.

La fidélité de $\mathcal{H}$ nous garantit que $[\![\mathcal{H}]\!]^B_S(\alpha(m)) \approx_{\beta^f_{\alpha(m)}} val^{\alpha(m)}_S$. En particulier $val^{\alpha(m)}_S(\pi) \geq_{\beta^f_{\alpha(m)}} [\![\mathcal{H}]\!]^B_S(\alpha(m))(\pi) \geq val_S(\tau) \geq n$.

Puisque $val_B(\pi) \leq \alpha(m)$, on a montré $val_S(\pi) \geq_{\beta^f_{\alpha(m)}} n$. C'est vrai pour toute branche π de $R_\mathcal{A}$, donc on obtient $val_B(R_\mathcal{A}) \leq \alpha(m)$ et $val_S(R_\mathcal{A}) \geq_{\beta^f_{\alpha(m)}} n = [\![\mathcal{C}]\!]^B_S(m)(t)$.

L'exécution $R_\mathcal{A}$ témoigne donc bien de l'inégalité (1) : $[\![\mathcal{A}]\!]^B_S(\alpha(m)) \preccurlyeq_{\beta_m} [\![\mathcal{C}]\!]^B_S(m)(t)$, avec $\beta^1_m = \beta^f_{\alpha(m)}$.

On montre maintenant la réciproque : l'inégalité (2).

Soit $t \in \mathcal{T}_\mathbb{A}$ un arbre infini, et $m \in \mathbb{N}$. Soit $n = [\![\mathcal{A}]\!]^B_S(m)(t)$ et $R_\mathcal{A}$ une exécution témoin. On veut montrer qu'il existe une exécution $R_\mathcal{C}$ de $\mathcal{C}$ sur t témoignant $[\![\mathcal{C}]\!]^B_S(\alpha(m))(t) \geq_{\beta^2_m} n$, pour un certain β^2_m.

Pour construire une telle exécution $R_\mathcal{C}$, on va combiner $R_\mathcal{A}$ avec l'une des stratégies hd qui permettent de diriger les exécutions de $\mathcal{H}$.

Soit k maximal tel que $\beta^f_m(\beta^{hd}_m(k)) \leq n$. On définit $R_\mathcal{C}$ en utilisant $R_\mathcal{A}$ pour les transitions de $\mathcal{A}$, et δ_k pour les transitions de $\mathcal{H}$.

Il faut montrer que $R_\mathcal{C}$ convient. Soit τ une branche de $R_\mathcal{C}$, on veut montrer $val_S(\tau) \geq_{\beta^2_m} n$, pour une certaine fonction β^2_m.

L'exécution τ est en fait le résultat de l'exécution R_k de $\mathcal{H}$ pilotée par δ_k, sur la branche π correspondante de $R_\mathcal{A}$.

Par définition de $R_\mathcal{A}$, $val_B(\pi) \leq m$ et $val_S(\pi) \geq n$. Par fidélité de $\mathcal{H}$, on a donc $val_B(\tau) \leq \alpha(m)$. Il nous reste donc à vérifier que le transducteur $\mathcal{H}$ guidé par la stratégie δ_k fonctionne correctement sur le mot π, avec $val_B(\pi) \leq m$ et $val_S(\pi) \geq n$.

En utilisant la fidélité de $\mathcal{H}$, mais pour la S-sémantique cette fois, on obtient $[\![\mathcal{H}]\!]_S(\pi) \geq_{\beta^f_m} n$, puisque $val_S(\pi) \leq m$.

Par définition de k, on a $\beta^f_m(\beta^{hd}_m(k)) \leq n \leq \beta^f_m([\![\mathcal{H}]\!]_S(\pi))$. Puisque β^f_m est choisie croissante, ceci nous permet d'obtenir $\beta^{hd}_m(k) \leq [\![\mathcal{H}]\!]_S(\pi)$, et donc par le Lemme 12.3.2, R_k est acceptante sur π. La branche résultante τ est donc bien infinie, et puisque la S-valeur de R_k est k, on obtient $val_S(\tau) = k$.

Par définition de k (incluant sa maximalité), on a $\beta^f_m(\beta^{hd}_m(k+1)) \geq n$. Cela signifie qu'en posant $\beta^2_m(x) = \beta^f_m(\beta^{hd}_m(x+1)$, on obtient $\beta^2_m(val_S(\tau)) \geq n$.

Ceci achève la preuve que $R_\mathcal{C}$ est bien un témoin de $[\![\mathcal{C}]\!]^B_S(\alpha(m))(t) \geq_{\beta^2_m} n$, pour la fonction de correction β^2_m décrite ci-dessus.

On peut finalement choisir $\beta_m = \max(\beta^1_m, \beta^2_m)$ pour conclure la démonstration du théorème.

La deuxième partie de l'énoncé est déduite du fait que dans toute la preuve, on a pu se limiter aux stratégies hd de $\mathcal{H}$. □

12.4 Transducteurs et hiérarchisation

Le but de cette section est de montrer que tout BS-NBA $\mathcal{A}$ peut être transformé en un hBS-NBA $\mathcal{A}'$, en composant $\mathcal{A}$ avec un hBS-T.

12.4.1 Hiérarchisation des B- et S-automates

On commence par montrer ce résultat sur les B- et S-automates.

On sait depuis [Col11] que les automates hiérarchiques suffisent dans les automates de coût. Dans [CL10], un transducteur hd qui transforme un mot $u \in \mathbb{C} = (\mathbb{C}_B^\Gamma)$ en $u \in \mathbb{C}_h$ de valeur équivalente est donné explicitement. Composer ce transducteur avec un B-NBA $\mathcal{A}$ nous permet donc d'obtenir un B-NBA $\mathcal{A}'$ hiérarchique, tel que $[\![\mathcal{A}]\!]_B \approx [\![\mathcal{A}']\!]_B$.

Le même résultat pour les S-automates était déjà connu par Colcombet et Löding, une construction explicite d'un tel transducteur hd est donnée par Vanden Boom dans [KV11].

On pourra donc supposer par la suite que les B-actions d'une part, et les S-actions d'autre part, sont hiérarchiques. Dans le contexte des BS-automates, ceci siginifie que l'on a un ordre de hiérarchie sur les B-compteurs Γ_B, et un ordre sur les S-compteurs Γ_S, mais pas d'ordre global sur $\Gamma = \Gamma_B \uplus \Gamma_S$.

12.4.2 Hiérarchisation des BS-automates

On veut maintenant obtenir une hiérarchisation globale sur les BS-T, c'est-à-dire que l'on veut montrer le théorème suivant :

Théorème 12.4.1 *Pour tous ensembles Γ_B, Γ_S de compteurs, il existe un hBS-T $\mathcal{H}(\Gamma_B, \Gamma_S)$ fidèle et déterministe en histoire sur l'aphabet $\mathbb{C} = \mathbb{C}_B^{|\Gamma_B|} \times \mathbb{C}_S^{|\Gamma_S|}$.*

On peut supposer que les actions sont B-hiérarchiques et S-hiérarchiques, en composant au besoin avec les transducteurs décrits dans la Section 12.4.1.

Puisque seuls comptent le nombre de B-compteurs et de S-compteurs, on écrira $\mathcal{H}_{K_B}^{K_S}$ au lieu de $\mathcal{H}(\Gamma_B, \Gamma_S)$ pour le hBS-T que l'on cherche à construire.

On commence par décrire les idées principales qui interviennent dans la construction de $\mathcal{H}_{K_B}^{K_S}$. Cet automate possèdera K_B B-compteurs et $(K_B+1)K_S$ S-compteurs. Le principe de cet automate est de découper le mot donné en entrée en séquences de S-actions de $\{\mathtt{i}, \varepsilon\}^*$, qui se produisent entre deux remises à zéro d'un B-compteur.

On utilisera deux S-compteurs de sortie pour chaque S-compteur d'origine : une pour compter le nombre d'incréments dans chaque séquence, et une autre pour compter le nombre de séquences qui contiennent au moins un incrément.

De cette manière, si la S-valeur était grande comparée à la B-valeur (que l'on bornera explicitement), l'une de ces copies atteindra forcément une grande valeur.

Cette intuition est déjà présente dans [BC06], mais elle est utilisée d'un point de vue booléen : on sépare juste le cas borné du cas non borné. On étend ici cette idée de manière à prendre en compte des propriétés quantitatives.

Par ailleurs, notre transducteur va copier presque exactement les B-actions données en entrée. La seule modification possible sera d'ajouter un $\mathtt{r}$ dans chaque intervalle séparé par deux $\mathtt{r}$, ce qui change la B-valeur par un facteur 2 au maximum. Ces remarques permettront de conclure que le transducteur est fidèle.

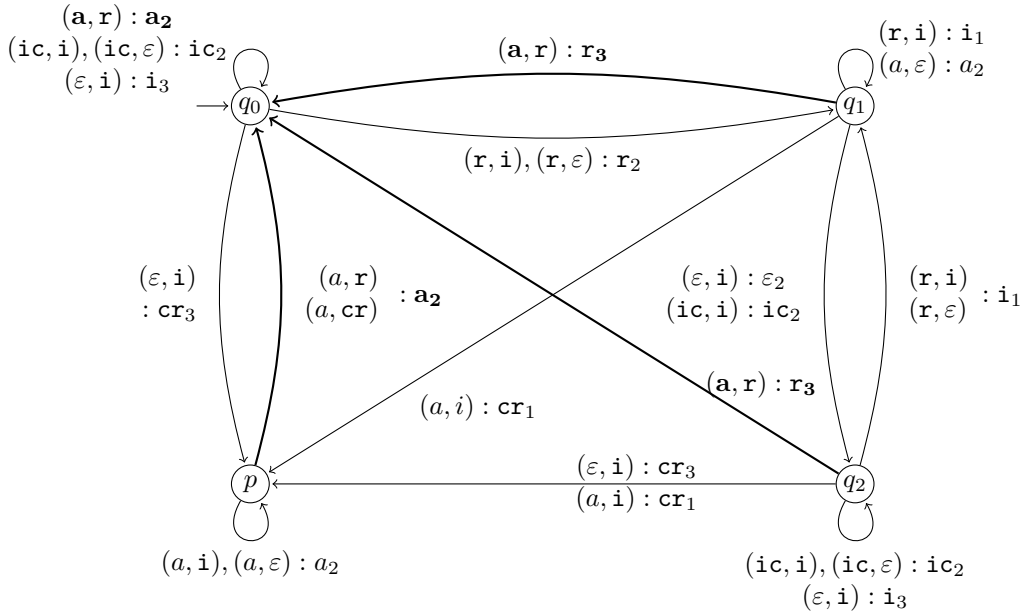
On montrera ensuite que le transducteur $\mathcal{H}^{K_S}_{K_B}$ ainsi construit est déterministe en histoire.

Cela signifie qu'en composant ce transducteur avec un automate alternant, on pourra obtenir un automate hiérarchique équivalent.

Construction de $\mathcal{H}^1_1$.

On commence par traiter le cas $K_B = K_S = 1$. On veut donc construire un transducteur lisant un mot infini sur $\mathbb{C} = \mathbb{C}_B \times \mathbb{C}_S$, et renvoyant un mot d'actions hiérarchiques, tout en garantissant la BS-équivalence entre l'entrée et la sortie. On représente ci-dessous l'automate $\mathcal{H}^1_1$. Les transitions étiquetées par $\mathbb{C}_B \times \{\mathtt{r}, \mathtt{cr}\}$ sont représentées en gras pour plus de lisibilité.

Cet automate possède 3 compteurs hiérarchiques S_1, B_2, S_3. Les actions hiérarchiques sur ces compteurs sont notées de manière abrégée : par exempe $\mathtt{ic}_2$ représente l'action $(\varepsilon, \mathtt{ic}, \mathtt{r})$ qui laisse S_1, inchangé, effectue $\mathtt{ic}$ sur B_2, et remet S_3 à zéro. On note a une action quelconque de $\mathbb{C}_B$. Pour tout état s, il y a une boucle $s \xrightarrow{(\varepsilon,\varepsilon):\varepsilon_1} s$ non représentée sur le schéma. De plus tous les états sont acceptants.



On commence par remarquer que $\mathcal{H}^1_1$ peut être vu comme un automate soit sur mots finis, soit sur mots infinis, car la seule manière de refuser un mot est de se retrouver sans transition à emprunter. On peut donc utiliser le fait que pour tout $w \in \mathbb{C}^\omega$, et toute exécution ρ de $\mathcal{H}^1_1$ sur w, on a $val_B(\rho) = \sup\{val_B(\rho') : \rho' \text{ préfixe de } \rho\}$, et pour tout $m \in \mathbb{N}$, on a également

$[\![\mathcal{H}_1^1]\!]_S^B(m)(w) = \inf\{[\![\mathcal{H}_1^1]\!]_S^B(m)(u) : u \text{ préfixe de } w\}$. La correction de $\mathcal{H}_1^1$ sur mots finis implique donc sa correction sur mots infinis.

On décrit maintenant le principe de cet automate. On appellera B-intervalle une séquence d'actions encadrée par deux B-resets, sans B-reset à l'intérieur. Sur une séquence de S-actions de $\{\mathtt{i}, \varepsilon\}$, le compteur γ_1 compte le nombre de B-intervalles contenant au moins un $\mathtt{i}$ un entrée, tandis que γ_3 compte le nombre de $\mathtt{i}$ à l'intérieur durant chaque ε-séquence du B-compteur d'entrée. Le fait que la valeur de ce B-compteur soit bornée nous garantit la correction de la BS-sémantique.

Théorème 12.4.2 *$\mathcal{H}_1^1$ est fidèle.*

Démonstration Il faut d'abord montrer que pour tout $u \in \mathbb{C}^\omega$ toute exécution ρ de $\mathcal{H}_1^1$ sur u vérifie $val_B(\rho) \approx_\alpha val_B(u)$.

Les seules transitions de $\mathcal{H}_1^1$ qui ne reproduisent pas l'action du B-compteur d'entrée sont les transitions en gras étiquetées $(a, r) : \mathtt{r}_3$, ainsi que celles dirigées vers p étiquetées $(a, \mathtt{i}) : \mathtt{cr}_1$. Toutes ces transitions effectuent un $\mathtt{r}$ sur le B-compteur γ_2, au lieu de l'action a lue en entrée.

Cependant, pour atteindre ces transitions depuis q_0, il faut être passé dans q_1, et donc avoir lu en entrée un reset pour le B-compteur. De plus, pour revoir une telle transition, il faut obligatoirement repasser par q_0. Cela signifie que l'on peut ajouter au plus un reset sur B_2 entre deux resets lu en entrée, possiblement en remplaçant un $\mathtt{ic}$.

Si ρ est une exécution sur un mot $u \in \mathbb{C}^\omega$, on obtient donc $val_B(\rho) \leq val_B(u) \leq 2 * val_B(\rho) + 1$, ce qui suffit à conclure $val_B(\rho) \approx_\alpha val_B(u)$, avec $\alpha(m) = 2m + 1$.

On montre maintenant la deuxième condition pour l'α-fidélité de $\mathcal{H}_1^1$.

Soit $m \in \mathbb{N}$ fixé, on veut montrer qu'il existe β_m tel que $[\![\mathcal{H}_1^1]\!]_S^B(m) \approx_{\beta_m} val_S^m$.

On doit donc montrer les deux inégalités suivantes

– $[\![\mathcal{H}_1^1]\!]_S^B(m) \preccurlyeq_{\beta_m} val_S^m$ (1)

– $val_S^m \preccurlyeq_{\beta_m} [\![\mathcal{H}_1^1]\!]_S^B(m)$ (2)

Pour la partie (1), on peut remarquer que nos S-compteurs S_1 et S_3 sont incrémentés seulement lorsqu'un incrément est lu dans le mot d'entrée u. De plus, il faut avoir atteint l'état p pour être autorisé à lire un $\mathtt{cr}$ en entrée. Cependant, le seul moyen d'atteindre cet état est d'effectuer un $\mathtt{cr}$ en sortie. Cela implique que tout $\mathtt{cr}$ de u peut être mis en relation avec un $\mathtt{cr}$ effectué en sortie un peu plus tôt, sur S_1 ou S_3. Une fois dans p, les S-incréments sont ignorés jusqu'au prochain S-reset, qui nous ramène dans q_0. Toutes les valeurs enregistrées en sortie sont donc inférieures à celles enregistrées en entrée.

On obtient $[\![\mathcal{H}_1^1]\!]_S^B(m) \leq val_S^m$.

Pour la partie (2), on suppose pour la contradiction qu'il existe un ensemble U de mots sur $\mathbb{C}$ tels que $\{[\![\mathcal{H}_1^1]\!]_S^B(m)(u) : u \in U\}$ est borné par un certain $N \in \mathbb{N}$, mais $\{val_S^m(u) : u \in U\}$ n'est pas borné.

Le fait que le premier ensemble soit borné force $U \subseteq L_m$: en effet, si $val_B(u) > m$, alors $[\![\mathcal{H}_1^1]\!]_S^B(m)(u) = \infty$. On sait donc que pour tout $u \in U$, $val_B(u) \leq m$.

Soit $u \in U$ tel que $val_S(u) > ((N+1)m+1)N$.

On écrit $u = (u_B, u_S)$, et on factorise u_S en $u_1 b_1 u_2 b_2 \dots u_k b_k u_{k+1}$ de manière à ce que pour tout j, $b_j \in \{\mathtt{r}, \mathtt{cr}\}$ et $u_j \in \{\mathtt{i}, \varepsilon\}^*$. Ceci induit une factorisation $u_B = v_1 a_1 v_2 a_2 \dots v_k a_k v_{k+1}$ en faisant correspondre les positions des a_j et des b_j.

Remarquons qu'une exécution ρ de $\mathcal{H}_1^1$ sur u peut s'écrire $\rho_1 \dots \rho_k \rho'$ avec pour tout j, ρ_j est la portion de ρ effectuée sur $(v_j a_j, u_j b_j)$. Par définition des transitions représentées en gras dans $\mathcal{H}_1^1$, pour tout j,ρ_j commence et finit en q_0.

De plus, soit $n = [\![\mathcal{H}_1^1]\!]_S(u) \leq N$, il existe une exécution optimale ρ de $\mathcal{H}_1^1$sur u avec $val_S(\rho) = n$. Ici « optimale » signifie que pour tout j, $val_S(\rho_j)$ est maximal : il n'existe aucune exécution partielle ρ'_j sur $(v_j a_j, u_j b_j)$, partant de q_0, et telle que $val_S(\rho'_j) > val_S(\rho_j)$. On peut toujours choisir ρ optimale car ses portions ρ_j sont indépendantes les unes des autres.

Soit J l'ensemble d'indices j tels que $b_j = \mathtt{cr}$. Pour tout $j \in J$, on a $val_S(u_j\mathtt{cr}) \geq val_S(u) = val_S^m(u) > ((N+1)m+1)N$. Soit $j \in J$ tel que $val_S(\rho_j) = n$, qui existe parce que $val_S(\rho) = n$. On a $q_0 \xrightarrow{(v_j,u_j)}{}^* p \xrightarrow{(b_j,\mathtt{cr})} q_0$.

Soit $v_j = w_0 \mathtt{r} w_1 \mathtt{r} \dots \mathtt{r} w_s$ avec $w_l \in \{\mathtt{ic}, \varepsilon\}^*$ pour tout l, et soit $u_j = x_0 d_0 x_1 \dots d_{s-1} x_s$ la factorisation correspondante de u_j, c'est-à-dire que pour tout l, $|x_l| = |w_l|$ et $|d_l| = 1$.

Si $s = 0$, alors la transition de q_0 à q_1 n'est pas prise dans ρ_j. Mais quand la transition vers p est prise, on effectue l'action $\mathtt{cr}_3$. Cela signifie que le compteur S_3 doit avoir une valeur au moins n, et donc que l'on a vu au moins n lettres $(\varepsilon, \mathtt{i})$ consécutives. Supposons qu'il y a au plus n lettres $(\varepsilon, \mathtt{i})$ consécutives dans (v_j, u_j). On sait que $|v_j|_{\mathtt{ic}} \leq m$ donc on obtient $|u_j|_{\mathtt{i}} \leq (n+1)m$. Or, on a dans ce cas $val_S(u_j\mathtt{cr}) > ((N+1)m+1)(\alpha(m)+1) > (n+1)m$, c'est absurde. Il y a donc strictement plus que n lettres $(\varepsilon, \mathtt{i})$ consécutives dans (v_j, u_j), ce qui contredit l'optimalité de ρ, puique l'on peut alors construire ρ'_j sur $(v_j a_j, u_j b_j)$ telle que $val_S(\rho'_j) > n = val_S(\rho_j)$, en attendant un peu plus longtemps avant d'aller dans l'état p.

Il reste à traiter le cas $s \geq 1$. Comme précédemment, le fait que ρ soit optimal implique $\sup_i(|x_i|_{\mathtt{i}}) \leq (n+1)m$. par conséquent, pour tout i, $|x_i d_i|_{\mathtt{i}}) \leq (n+1)m+1$. On peut remarquer que la valeur de S_1 après la lecture de (w_i, x_i) est égale à la taille de $J_i = \{l \leq i, |x_l d_l|_{\mathtt{i}} > 0\}$. On doit donc avoir $|J_s| \leq n$ par optimalité de ρ : si ce n'était pas le cas on pourrait construire une exécution de S-valeur supérieure à celle de ρ_j, en choisissant une transition vers p étiquetée $\mathtt{cr}_3$ à la fin de (w_j, x_j).

Finalement, $|u_j|_{\mathtt{i}} \leq (|x_j d_j|_{\mathtt{i}})|J_s| \leq ((n+1)m+1)n \leq ((N+1)m+1)N$. On a donc $val_S(u_j\mathtt{cr}) \leq ((N+1)m+1)N$. Ceci est vrai pour tout j, $val_S(u) \leq ((N+1)m+1)N$, on obtient donc une contradiction avec la supposition initiale $val_S(u) > ((N+1)m+1)N$.

On peut en conclure qu'un tel ensemble U n'existe pas, et $val_S^m \preccurlyeq_{\beta_m} [\![\mathcal{H}_1^1]\!]_S$, pour $\beta_m(n) = ((n+1)m+1)n$. □

Théorème 12.4.3 *$\mathcal{H}_1^1$ est déterministe en histoire.*

Démonstration On a déjà montré que la B-valeur d'un mot est indépendante de l'exécution choisie, à α près, avec $\alpha(m) = 2m + 1$.

On peut remarquer de plus que le seul choix non-déterministe dans $\mathcal{H}_1^1$ se produit lorsque l'on lit une S-action $\mathtt{i}$ en entrée : on a alors le choix entre rester dans l'état courant et aller dans l'état p, si l'on ne s'y trouve pas déjà.

On définit la stratégie δ_n comme faisant le choix d'aller dans p avec une transition étiquetée $\mathtt{cr}_i$ si la valeur n est atteinte sur le S-compteur S_i (la valeur des compteurs ne dépend que de l'historique de l'éxécution). Cette stratégie définit pour tout mot $u \in \mathbb{C}^\omega$ une unique exécution ρ_n de $\mathcal{H}_1^1$, telle que $val_S(\rho_n) \geq n$. En effet, chaque action $\mathtt{cr}$ est effectuée sur un compteur qui a atteint la valeur n.

De plus, s'il existe une exécution infinie ρ de $\mathcal{H}_1^1$ sur u telle que $val_S(\rho) \geq n$, alors l'exécution dirigée par δ_n est également infinie : elle correspond juste à prendre les transitions vers p plus tôt que dans ρ.

Ceci conclut le preuve que $\mathcal{H}_1^1$ est déterministe en histoire. □

Extension de $\mathcal{H}_1^1$ à plusieurs compteurs.

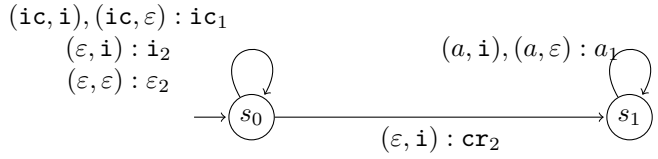
Soit $K_B, K_S > 0$, on définit $\mathbb{C}_{K_B}^{K_S} \subset (\mathbb{C}_B)^{K_B} \times (\mathbb{C}_S)^{K_S}$ l'alphabet des actions qui sont B-hiérarchiques sur K_B B-compteurs et S-hiérarchiques sur K_S S-compteurs. Autrement dit, les lettres de $\mathbb{C}_{K_B}^{K_S}$ sont toutes de la forme $((\mathtt{r}, \ldots, \mathtt{r}, a, \varepsilon, \ldots, \varepsilon), (\mathtt{r}, \ldots, \mathtt{r}, b, \varepsilon, \ldots, \varepsilon))$ avec $(a, b) \in \mathbb{C}_B \times \mathbb{C}_S$.

Pour tous $K_B, K_S > 0$, on veut construire $\mathcal{H}_{K_B}^{K_S}$ qui généralise $\mathcal{H}_1^1$ à l'alphabet $\mathbb{C}_{K_B}^{K_S}$. Comme précédément, le but est d'obtenir $\mathcal{H}_{K_B}^{K_S}$ hiérarchique, fidèle, et déterministe en histoire. On commence par définir par récurrence les automates $\mathcal{H}_{K_B}^1$ pour $K_B > 0$.

Construction de $\mathcal{H}_{K_B}^1$ pour tout $K_B > 0$.

On va partir de $\mathcal{H}_1^1 = \langle \mathbb{C}_1^1, Q^1, \{q_0^1\}, \Gamma_B^1, \Gamma_S^1, \Delta^1 \rangle$, que l'on a prouvé correct pour tous les critères que l'on s'est fixés.

Supposons maintenant que $\mathcal{H}_{K_B}^1 = \langle \mathbb{C}_{K_B}^1, Q^{K_B}, \left\{q_0^{K_B}\right\}, \Gamma_B^{K_B}, \Gamma_S^{K_B}, \Delta^{K_B} \rangle$ est construit et correct. On veut construire $\mathcal{H}_{K_B+1}^1$. On utilisera un BS-T auxiliaire $\mathcal{C} = \langle \{\mathtt{ic}, \varepsilon\} \times \{\mathtt{i}, \varepsilon\}, Q_\mathcal{C}, \{s_0\}, \{\gamma_B\}, \{\gamma_S\}, \Delta_\mathcal{C} \rangle$, à deux compteurs hiérarchiques $\gamma_B > \gamma_S$, défini par ce schéma :



Soit $\mathcal{H}^1_{K_B+1} = \langle \mathbb{C}^1_{K_B+1}, Q^{K_B+1}, \left\{q_0^{K_B+1}\right\}, \Gamma_B^{K_B+1}, \Gamma_S^{K_B+1}, \Delta^{K_B+1}\rangle$, que l'on va définir à l'aide de $\mathcal{H}^1_{K_B}$ et $\mathcal{C}$. On prend $Q^{K_B+1} = Q^{K_B} \times Q_{\mathcal{C}}$, $q_0^{K_B+1} = (q_0^{K_B}, s_0)$. On définit également $\Gamma_B^{K_B+1} = \Gamma_B^{K_B} \cup \{\gamma_B\}$ et $\Gamma_S^{K_B+1} = \Gamma_S^{K_B} \cup \{\gamma_S\}$, ordonnés par $\gamma > \gamma_B > \gamma_S$ pour tout $\gamma \in \Gamma_B^{K_B} \cup \Gamma_S^{K_B}$. La hiérarchie des compteurs est héritée de $\mathcal{H}^1_{K_B}$ et $\mathcal{C}$, et toute action sur les compteurs provenant de $\mathcal{H}^1_{K_B}$ effectue un reset sur tous les compteurs de $\mathcal{C}$.

Finalement, la table de transitions Δ^{K_B+1} est définie comme suit :

(C) pour tout $s \xrightarrow{(a,b):\sigma} s'$ dans $\Delta_{\mathcal{C}}$ avec $a \in \mathbb{C}_B$ et $b \in \mathbb{C}_S$, et tout $q \in Q^{K_B}$, on ajoute une transition $(q,s) \xrightarrow{(\varepsilon_{K_B} a,b):\sigma} (q,s')$ dans Δ^{K_B+1}.

(Hx) pour tout $q \xrightarrow{(a,b):\sigma} q'$ dans Δ^{K_B} avec $a \in \mathbb{C}_B^{K_B}$ et $b \in \mathbb{C}_S^1$, et tout $s \in Q_{\mathcal{C}}$, on ajoute une transition $(q,s) \xrightarrow{(a\mathtt{r},b):\sigma} (q', s_0)$ dans Δ^{K_B+1}.

(s1) pour tout $q \in Q^{K_B}$, et $a \in \mathbb{C}_B^{K_B+1}$, on ajoute une transition $(q,s_1) \xrightarrow{(a,\mathtt{cr}):a} (q_0^{K_B}, s_0)$ dans Δ^{K_B+1}.

Ici $\varepsilon_{K_B} a$ est une notation pour $(\varepsilon, \ldots, \varepsilon, a)$, où $a \in \mathbb{C}_B$. L'action σ utilisée est effectuée sur les mêmes compteurs qu'à l'origine, c'est-à-dire $\{\gamma - B, \gamma_S\}$ dans le cas (C), et $\Gamma_B^{K_B} \cup \Gamma_S^{K_S}$ dans le cas (Hx).

On montre maintenant que $\mathcal{H}^1_{K_B+1}$ est fidèle.

Par induction, on peut montrer en observant l'automate ainsi construit que comme dans le cas de $\mathcal{H}^1_1$, pour chaque B-compteur, on peut ajouter au plus un nouveau $\mathtt{r}$ entre deux $\mathtt{r}$ du mot d'entrée, et ce sont les seules modifications effectuées sur les B-actions.

On obtient donc le premier résultat voulu : si ρ est une exécution de $\mathcal{H}^1_{K_B+1}$ sur u, alors $val_B(\rho) \approx_\alpha val_B(u)$.

On doit maintenant montrer que pour tout m, il existe β_m tel que :

– $[\![\mathcal{H}^1_{K_B+1}]\!]^B_S(m) \preccurlyeq_{\beta_m} val^m_S$ (1)

– $val^m_S \preccurlyeq_{\beta_m} [\![\mathcal{H}^1_{K_B+1}]\!]^B_S(m)$ (2)

La partie (1) est similaire à celle de la section précédente : les incréments de sortie correspondent toujours à des incréments en entrée, et chaque $\mathtt{cr}$ en entrée induit un $\mathtt{cr}$ en sortie, effectué après autant ou moins d'incréments.

Il reste à prouver la partie (2). Soit $m \in \mathbb{N}$. Soit $u = (u_B, u_S) \in (\mathbb{C}_B^{K_B+1} \times \{\mathtt{i}, \varepsilon\})^*(\mathbb{C}_B^{K_B+1} \times \{\mathtt{cr}\})$ tel que $val^m_S(u) > N$ et $[\![\mathcal{H}^1_{K_B+1}]\!]^B_S(m)(u) \leq n$. On restreint u à ce langage, car comme dans le cas de $\mathcal{H}^1_1$, l'automate revient dans son état initial quand il lit un $\mathtt{cr}$, et les exécutions sur ces facteurs sont indépendantes les unes des autres. Remettre à zéro le S-compteur au moyen d'un $\mathtt{r}$ correspond à ignorer u : l'automate peut alors éviter les transitions effectuant un $\mathtt{cr}$, et revenir à l'état p après avoir lu u.

On écrit $u_B = u_1 v_1 u_2 \ldots u_k v_k a$ où pour tout $i \leq k$, $u_i \in (\mathbb{C}_B^{K_B} \times \{\mathtt{r}\})^*$ et $v_i \in (\{\varepsilon\}^{K_B} \times \{\mathtt{ic}, \varepsilon\})^*$. Pour l'unicité de la factorisation, seulement u_1 et v_k peuvent valoir ε. a est juste la dernière lettre de u_B. On effectue la factorisation correspondante $u_S = u'_1 v'_1 u'_2 \ldots u'_k v'_k \mathtt{cr}$.

Soit ρ une exécution de $\mathcal{H}^1_{K_B+1}$ sur u. On peut écrire $\rho = \rho_1 \tau_1 \rho_2 \ldots \tau_{k-1} \rho_k$ où $\rho_{K_B} = \rho_1 \ldots \rho_k$ est une exécution de $\mathcal{H}^1_{K_B}$ sur $u_1 \ldots u_k$, et pour tout $i < k$,

τ_i est une exécution de $\mathcal{C}$ sur v_i.

Par hypothèse de récurrence, il existe une fonction de correction $\beta^m_{K_B}$ tel que $|u_1 \dots u_k|_{\mathtt{i}} \leq \beta^m_{K_B}(val_S(\rho_{K_B}))$. Soit val'_S la fonction qui à une exécution associe la valeur maximale atteinte par un S-compteur, non nécessairement checkée Pour $\beta^m_{\mathcal{C}}(n) = (m+1)n$, on a pour tout $i < k$, $|u_i|_{\mathtt{i}} \leq_{\beta^m_{\mathcal{C}}} val'_S(\tau_i)$.

On obtient $val_S(u) \leq_{\beta^m_{K_B+1}} val_S(\rho)$, pour $\beta^m_{K_B+1} = \max(\beta^m_{K_B}, \beta^m_{\mathcal{C}})$, ce qui nous donne le résultat voulu.

On a donc construit par récurrence, pour tout $K_B > 0$, un hBS-T $\mathcal{H}^1_{K_B}$ fidèle. Le déterminisme en histoire de $\mathcal{H}^1_{K_B}$ est obtenu par une extension canonique des stratégies décrites plus haut : on prend les transitions étiquetée `cr` dès que la valeur n est atteinte sur le S-compteur correspondant.

Par récurrence, pour tout $K_B > 0$, l'automate $\mathcal{H}^1_{K_B}$ a 2^{K_B+1} états et $2K_B+1$ compteurs.

Construction de $\mathcal{H}^{K_S}_{K_B}$ pour tout $K_B > 0$ et $K_S > 0$.

On fixe des valeurs K_B et K_S, et on décrit l'automate $\mathcal{H}^{K_S}_{K_B}$.

Soit $\mathcal{H}^1_{K_B}$ tel que défini dans la section précédente. Ces compteurs sont $S_{K_B+1}B_{K_B}S_{K_B} \dots B_1S_1$.

$\mathcal{H}^{K_S}_{K_B}$ a les mêmes B-compteurs que $\mathcal{H}^1_{K_B}$, c'est-à-dire $(\gamma_i)_{i\in[1,K_B]}$. Ces S-compteurs sont $(\gamma^j_i)_{i\in[1,K_B+1],j\in[1,K_S]}$.

On définit l'ordre hiérarchique $\gamma^j_i < \gamma^{j'}_{i'}$ par l'ordre lexicographique sur (i,j), et de plus pour tous i,j, $\gamma_i > \gamma^j_i$. Finalement, pour tout j, $\gamma^j_{K_B+1} > \gamma_i$.

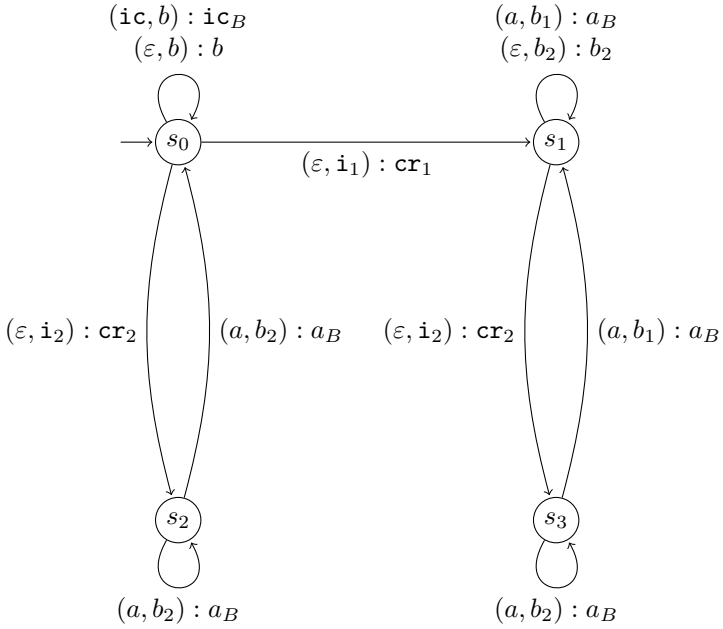
Soit γ_i un B-compteur de $\mathcal{H}^{K_S}_{K_B}$.

Le principe de $\mathcal{H}^1_{K_B}$ est de compter le nombre d'incréments de S-compteurs inférieurs, durant les séquences de ε de γ_i, et d'effectuer un `cr` quand cette valeur devient assez grande (se dirigeant ainsi vers un état où la lecture d'un `cr` est autorisée). Le S-compteur le plus haut dans la hiérarchie compte le nombre de séquences de $\{\mathtt{ic}, \varepsilon\}$ de γ_{K_B}, précédées d'un `r` pour γ_{K_B}, et qui contiennent un `i` pour le S-compteur.

On a dupliqué ici chaque S-compteur γ'_i en K_S compteurs γ^j_i qui jouent le même rôle, mais relativement au compteur d'origine γ_j.

Les K_S S-compteurs d'entrée sont hiérarchiques, ce qui induit que la condition hiérarchique globale est respectée par $\mathcal{H}^{K_S}_{K_B}$.

On décrit par exemple ici l'automate $\mathcal{C}_2$ qui joue le rôle de $\mathcal{C}$ dans le cas $K_S = 2$. Cet automate est défini sur l'alphabet $\{\mathtt{ic}, \varepsilon\} \times \{\mathtt{i}_1, \mathtt{i}_2, \varepsilon_2\}$, et possède deux S-compteurs γ_1 et γ_2, et un B-compteur γ_B, ordonnés par $\gamma_B > \gamma_1 > \gamma_2$.



L'automate est autorisé à lire un `cr` pour γ_2 dans les états s_2 et s_3, et un `cr` pour γ_1 dans s_1 et s_3.

On peut montrer que $\mathcal{H}_{K_B}^{K_S}$ est un hBS-T fidèle et déterministe en histoire, en combinant les idées vues dans dans les preuves précédentes. L'automate $\mathcal{H}_{K_B}^{K_S}$ possède K_B B-compteurs, et $(K_B + 1)K_S$ S-compteurs.

12.5 Conclusion

En combinant les résultats précédents, on obtient le théorème suivant :

Théorème 12.5.1 *Soit $\mathcal{A}$ un BS-NBA. On peut construire un hBS-NBA $\mathcal{A}' = \mathcal{H} \circ \mathcal{A}$ tel que $\mathcal{A}' \cong \mathcal{A}$, où $\mathcal{H}$ est un BS-T hd fidèle.*

Démonstration Soit Γ_B, Γ_S les ensembles de compteurs de $\mathcal{A}$. Soit $\mathcal{H} = \mathcal{H}(\Gamma_B, \Gamma_S)$ le hBS-T obtenu par le Théorème 12.4.1.

Par le Théorème 12.3.3, le BS-NBA $\mathcal{A}' = \mathcal{H} \circ \mathcal{A}$ est BS-équivalent à $\mathcal{A}$. De plus, $\mathcal{H}$ est hiérarchique donc $\mathcal{A}'$ l'est également : les actions effectuées par $\mathcal{A}'$ sont données par $\mathcal{H}$. □

Chapitre 13

Théorème de caractérisation de Rabin

13.1 Version classique

13.1.1 Introduction du problème

Le but de cette section est de présenter l'équivalence classique suivante :

Théorème 13.1.1 ([Rab70, KV99]) *Soit L un langage d'arbres infinis. Les propriétés suivantes sont équivalentes :*

(1) L est reconnu par une formule de WMSO,
(2) L est reconnu par un automate faible alternant,
(3) L et son complément sont tous deux reconnus par des automates de Büchi non-déterministes.

Ce théorème a d'abord été montré par Rabin sans l'équivalence (2) [Rab70]. Kupferman et Vardi [KV99] ont ensuite démontré une construction purement automates pour la direction (3) $\Rightarrow$ (2), induisant une augmentation seulement quadratique du nombre d'états.

Tous les automates classiques mentionnés ici peuvent être vus comme des automates de coût sans compteurs, on ne les redéfinira donc pas. De tels automates définissent toujours des fonctions de la forme χ_L, où L est le langage reconnu par l'automate vu comme un automate classique. On identifiera toujours L et χ_L pour plus de commodité.

On va résumer les idées de la preuve de [KV99], que l'on désire étendre aux fonctions de coût. Il est nécessaire de bien comprendre cette preuve pour appréhender l'extension que l'on en fera, car on réutilise les mêmes idées de base, mais en y ajoutant tout une machinerie propre aux automates de coût.

On notera NBA pour « automate de Büchi non-déterministe », et WAA pour « automate faible alternant ».

On se donne donc deux NBA $\mathcal{U} = \langle Q, \mathbb{A}, In, F, \Delta \rangle$ et $\mathcal{U}' = \langle Q', \mathbb{A}, In', F', \Delta' \rangle$, tels que $[\![\mathcal{U}']\!] = \overline{[\![\mathcal{U}']\!]}$. Soit $L = [\![\mathcal{U}]\!]$, on cherche à construire un WAA W tel que $[\![W]\!] = L$.

13.1.2 Pièges

Afin d'analyser deux exécutions simultanées de $\mathcal{U}$ et $\mathcal{U}'$, on va définir la notion de piège pour deux NBA. Soit $m = |Q| \cdot |Q'|$. Une *frontière* E est un ensemble de noeuds de $\mathcal{T}$ tel que pour chaque branche π de $\mathcal{T}$, $E \cap \pi$ est un singleton. Par exemple pour tout $n \in \mathbb{N}$, $\{0,1\}^n$ est une frontière. Si E_i est une frontière et π est une branche, on notera e_i^π le seul noeud dans $E \cap \pi$.

Un *piège* pour $\mathcal{U}$ et $\mathcal{U}'$ est une séquence de frontières $E_0 = \{\epsilon\}, E_1, \ldots, E_m$ telle qu'il existe un arbre $t \in \mathcal{T}_\mathbb{A}$, et des exécutions R et R' de $\mathcal{U}$ et $\mathcal{U}'$ sur t satisfaisant la propriété suivante : pour tout $i \in [0, m[$ et toute branche π de t, il existe $x, x' \in [e_i^\pi, e_{i+1}^\pi[$ tels que $R(x) \in F$ et $R'(x') \in F'$. Chaque ensemble de positions $[e_i^\pi, e_{i+1}^\pi[$ sera appelé *bloc*. La condition stipule que chaque bloc doit contenir au moins un état acceptant de $\mathcal{U}$ et au moins un état acceptant de $\mathcal{U}'$.

Lemme 13.1.2 ([Rab70]) *S'il existe un piège pour deux NBA $\mathcal{U}$ et $\mathcal{U}'$, alors $[\![\mathcal{U}]\!] \cap [\![\mathcal{U}']\!] \neq \emptyset$.*

Ce lemme peut être montré par un argument de pompage : en réitérant à l'infini certains blocs, on peut construire un arbre accepté par les deux automates $\mathcal{U}$ et $\mathcal{U}'$.

13.1.3 Construction du WAA W

Kupferman et Vardi utilisent cette notion de piège pour construire directement un WAA W reconnaissant L. On donne ici les idées de la construction, voir [KV99] pour les définitions formelles.

L'automate W est le produit des automates $\mathcal{U}$ et $\mathcal{U}'$, ainsi qu'un automate qui permet de reconnaître les blocs, et de les compter jusqu'à m. Le contrôle des transitions de $\mathcal{U}$ est donné à Eve, tandis que celui de $\mathcal{U}'$ est donné à Adam. Les blocs sont mis ensuite à jour de manière déterministe : l'automate W incrémente son compteur à chaque fois qu'un bloc valide (i.e. contenant des états de F et de F') a été vu. Plus précisément, à chaque nouveau bloc, W va passer par les états suivants :

- $\bot$: Attente d'un état de F, état refusant
- $\top$: Attente d'un état de F', état acceptant

Lorsque l'on voit un état de F' dans $\top$, le compteur de bloc incrémente, et l'on repasse à $\bot$. Lorsque l'on termine le bloc numéro m, le compteur n'incrémente plus, et la transition mène vers `true` : Eve a gagné l'exécution et l'arbre est accepté.

De plus, Adam possède le contrôle du branchement, comme c'est déjà le cas dans un automate non-déterministe.

L'automate W est bien un B-WAA car on peut partitionner les états par rapport à la valeur du compteur dans $[0, m]$ ainsi que par la valeur $\bot$ ou $\top$. On obtient ainsi $2(m+1)$ partitions, chacune étant purement acceptante ou refusante, et telles que tout cycle de transitions est interne à une partition.

13.1.4 Correction de W

On analyse maintenant le comportement de W, et on montre que $[\![W]\!] = L$. On montre d'abord $L \subseteq [\![W]\!]$. Soit $t \in L$, et soit R une exécution acceptante de $\mathcal{U}$ sur t. Alors jouer R dans W est une stratégie gagnante pour Eve : peu importe les choix d'Adampour les transitions de $\mathcal{U}'$ et la branche π de t, il y aura une infinité d'états de F sur π, et l'automate W ne peut donc pas stabiliser dans l'état $\bot$, qui correspond à attendre un état de F. On a donc bien $t \in [\![W]\!]$, puisque Eve a une stratégie pour accepter t. On peut remarquer qu'on a pas utilisé l'hypothèse $[\![\mathcal{U}]\!] = \overline{[\![\mathcal{U}']\!]}$, et que la propriété $[\![\mathcal{U}]\!] \subseteq [\![W]\!]$ est en fait vérifiée même si W est défini à partir d'un automate $\mathcal{U}'$ quelconque.

Réciproquement, on veut montrer $[\![W]\!] \subseteq L$. Soit $t \notin L$, alors t est accepté par $\mathcal{U}'$. Soit R' une exécution acceptante de $\mathcal{U}'$ sur t. Supposons maintenant que $t \in [\![W]\!]$, et qu'il existe donc une stratégie gagnante σ pour Eve dans W qui permettrait d'accepter t. Cette stratégie induit une exécution R de $\mathcal{U}$ sur t.

Le fait que R' soit acceptante montre que sur toutes les branches, la seule manière de gagner est de dépasser les m blocs, puisqu'on ne peut pas stabiliser dans l'état $\bot$. On en conclut que les exécutions R et R' témoignent de l'existence d'un piège pour $\mathcal{U}$ et $\mathcal{U}'$. Le Lemme 13.1.2 nous donne alors une contradiction avec $[\![\mathcal{U}]\!] \cap [\![\mathcal{U}']\!] = \emptyset$. On a donc $t \notin [\![W]\!]$. On a montré $\overline{L} \subseteq \overline{[\![W]\!]}$, ce qui est équivalent à $[\![W]\!] \subseteq L$.

Finalement, l'automate W reconnait bien L. Son ensemble d'états Q_W est $Q \times Q' \times [0, m] \times \{\bot, \top\}$, on a donc $|Q_W| = \Theta(m^2)$, avec $m = |Q| \times |Q'|$.

13.2 Extension du Théorème de Rabin aux fonctions de coût

On souhaite maintenant étendre ce résultat sur les langages d'arbres infinis aux fonctions de coût sur les arbres infinis.

13.2.1 Quelles extensions choisir ?

Premièrement, la notation de complémentation n'a plus de sens pour les fonctions de coût. Cependant, on a vu que changer entre les sémantiques B et S peut être vu comme une complémentation, et reflète exactement la complémentation sur les fonctions de coût χ_L.

On remplace donc la condition « L et $\overline{L}$ sont reconnus par des NBA » par « f est reconnu par un B-NBA et par un S-NBA ».

On doit ensuite trouver la bonne extension pour les WAA. La première tentative, encouragée par les Théorèmes 11.3.1 et 11.3.3, a été d'utiliser les B-

WAA. Cependant, il s'avère que le résultat correspondant est faux : l'extension des WAA qui généralise le Théorème de Rabin est le modèle des B-QWAA.

On veut donc montrer le théorème suivant :

Théorème 13.2.1 *Soit f une fonction de coût sur abres infinis. On suppose qu'il existe un B-NBA $\mathcal{U}$ et un S-NBA $\mathcal{U}'$ tels que $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S \approx f$. Alors il existe un B-QWAA W tel que $[\![W]\!] \approx f$.*

Notons que ce Théorème est la réciproque du Théorème 11.3.4. La réunion de ces deux théorèmes nous donne donc bien une caractérisation de la classe des fonctions de coût reconnues par un B-QWAA.

13.2.2 Pièges de coût

On commence par étendre la notion de piège, de façon à les définir sur les automates de coût. Soit $\mathcal{U} = \langle Q_\mathcal{U}, \mathbb{A}, q_0^\mathcal{U}, F_B^\mathcal{U}, \Gamma_B^\mathcal{U}, \Delta_\mathcal{U}\rangle$ un B-NBA, et $\mathcal{U}' = \langle Q_{\mathcal{U}'}, \mathbb{A}, q_0^{\mathcal{U}'}, F_S^{\mathcal{U}'}, \Gamma_S^{\mathcal{U}'}, \Delta_{\mathcal{U}'}\rangle$ un S-NBA.

On veut définir la notion de « piège de coût » pour $\mathcal{U}$ et $\mathcal{U}'$. Un tel piège doit être incompatible avec $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$. Plus spécifiquement, ce piège va être le témoin d'une famille d'arbres possédant une B-valeur bornée pour $\mathcal{U}$ mais une S-valeur non bornée pour $\mathcal{U}'$. C'est donc un témoin de $[\![\mathcal{U}']\!]_S \not\preccurlyeq [\![\mathcal{U}]\!]_B$.

Or, définir de tels blocs est très difficile si les B-actions et S-actions de $\mathcal{U}$ et $\mathcal{U}'$ sont effectuées indépendamment. En effet, les multiples intéractions possibles des B- et S-actions seraient alors incontrôlables.

Soit $\mathcal{U} \times \mathcal{U}'$ le BS-NBA obtenu en faisant le produit de $\mathcal{U}$ et $\mathcal{U}'$.

Plus précisément, $\mathcal{U} \times \mathcal{U}' = \langle Q_\mathcal{U} \times Q_{\mathcal{U}'}, \mathbb{A}, (q_0^\mathcal{U}, q_0^{\mathcal{U}'}), F_B^\mathcal{U}, F_S^{\mathcal{U}'}, \Gamma_B^\mathcal{U}, \Gamma_S^{\mathcal{U}'}, \Delta_{\mathcal{U}\times\mathcal{U}'}\rangle$.

Afin d'obtenir des BS-actions hiérarchiques, on va utiliser le Théorème 12.5.1 pour obtenir un hBS-NBA $\mathcal{A} = \langle Q_\mathcal{A}, \mathbb{A}, q_0^\mathcal{A}, F_B, F_S, \Gamma_B, \Gamma_S, \Delta_\mathcal{A}\rangle$ tel que $\mathcal{A} \cong \mathcal{U} \times \mathcal{U}'$.

On rappelle que $\mathcal{A}$ est obtenu en composant $\mathcal{U} \times \mathcal{U}'$ avec un BS-T $\mathcal{H}$. De plus, on rappelle que $\mathcal{A}$ étant un automate d'arbres non-déterministe, une exécution de $\mathcal{A}$ est un arbre étiqueté par $\mathbb{C} \times Q_\mathcal{A}$ (sauf la racine étiquetée seulement par $q_\mathcal{A}^0$), où $\mathbb{C} = \mathbb{C}_B^{\gamma_B} \times \mathbb{C}_S^{\Gamma_S}$.

On utilisera $\mathcal{A}$ au lieu de $\mathcal{U} \times \mathcal{U}'$ pour définir le piège de coût, en raison de sa structure simplifiée de BS-actions.

Un *bloc* de $\mathcal{A}$ est une portion de branche d'exécution qui contient des états de F_B et de F_S (correspondant aux états acceptants de $\mathcal{U}$ et $\mathcal{U}'$ dans le piège classique), mais un bloc ne peut pas contenir un incrément d'un B-compteur $\gamma \in \Gamma_B$ s'il ne contient pas de reset pour γ. Cette seconde condition permet d'itérer un bloc sans provoquer d'inflation de la B-valeur.

Le nombre de blocs consécutifs requis est défini par $m := (|Q_\mathcal{A}| + 2)^{|\Gamma_S|+1}$.

Un *piège de coût* pour $\mathcal{A} = \mathcal{H} \circ (\mathcal{U} \times \mathcal{U}')$ consiste en une frontière E_m, et pour toute branche π une séquence de noeuds $e_0 = \{\epsilon\}, e_1^\pi, \ldots, e_m^\pi \in E_m$ sur π, ainsi qu'un arbre t et une exécution R de $\mathcal{A}$ sur t, vérifiant

– $val_S(R) > |Q_\mathcal{A}|$

– pour tout $0 \leq i < m$ et pour toute branche π de R, le segment $[e_i^\pi, e_{i+1}^\pi)$ est un bloc.
– si des branches π_1 et π_2 ont un préfixe en commun jusqu'à la position y, et $x < y$ est la première position telle que $e_i^{\pi_1} = x$ et $e_i^{\pi_2} \neq x$ pour un certain i, alors $e_i^{\pi_2} > y$. Cette condition permet d'itérer des blocs de π_2 sans endommager π_1.

Lemme 13.2.2 *Soit $\mathcal{U}$ un B-NBA et $\mathcal{U}'$ un S-NBA. Soit $\mathcal{A}$ un hBS-NBA tel que $\mathcal{A} \approxeq \mathcal{U} \times \mathcal{U}'$. S'il existe un piège de coût pour $\mathcal{A}$, alors $[\![\mathcal{U}']\!] \not\preccurlyeq [\![\mathcal{U}]\!]$.*

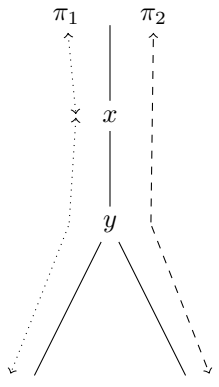
Démonstration

Soit $\mathcal{U}$ un B-NBA et $\mathcal{U}'$ un S-NBA. Soit $\mathcal{A} = \langle Q_\mathcal{A}, \mathbb{A}, q_0, F_B, F_S, \Gamma_B, \Gamma_S, \delta \rangle$ un hBS-NBA tel que $\mathcal{A} \approxeq \mathcal{U} \times \mathcal{U}'$. On suppose qu'il existe un piège de coût pour $\mathcal{A}$, et on veut montrer $[\![\mathcal{U}']\!] \not\preccurlyeq [\![\mathcal{U}]\!]$.

Soit $m := (|Q_\mathcal{A}| + 2)^{|\Gamma_S|+1}$. La définition du piège de coût pour $\mathcal{A}$ nous donne l'existence d'une frontière E_m, un ensemble de positions $e_0^\pi, \ldots, e_\pi^m$ pour chaque branche π, un arbre $t \in \mathcal{T}_\mathbb{A}$ et enfin une exécution R de $\mathcal{A}$ sur t.

Par définition des pièges de coût, on a $val_S(R) > |Q_\mathcal{A}|$, et pour toute branche π, $e_0^\pi < \cdots < e_m^\pi \in E_m$ sont des positions sur π telles que pour tout $0 \leq i < m$, la portion de R correspondant au segement $[e_i^\pi, e_{i+1}^\pi)$ est un bloc. On rappelle qu'un bloc est une portion d'exécution contenant des état de F_B et de F_S, et que pour tout compteur $\gamma \in \Gamma_B$, si γ est incrémenté dans cette portion alors il est aussi remis à zéro.

De plus, si des branches π_1 et $\pi_2$se séparent en position y et $x < y$ est la première position telle qu'il existe i avec $e_i^{\pi_1} = x$ et $e_i^{\pi_2} \neq x$, alors $e_i^{\pi_2} > y$ (on appellera ceci la *condition d'imbrication*). Dans ce cas, on dirait que $\pi_1 <_{imb} \pi_2$ et que y est un *noeud d'imbrication*. Si π_1 et π_2 partagent les e_i sur leur préfixe commun, on notera $\pi_1 \approx_{imb} \pi_2$. L'idée derrière ces définitions est la suivante : si $\pi_1 <_{imb} \pi_2$, alors itérer des blocs de π_2 n'endommagera pas les blocs de π_1.



La permière étape consiste à construire un arbre t' tel qu'il existe $n \in \mathbb{N}$ avec $[\![\mathcal{A}]\!]_B(t') \leq n < \infty$, et $[\![\mathcal{A}]\!]_S^B(n)(t') > |Q_\mathcal{A}|$ (on rappelle que $[\![\mathcal{A}]\!]_S^B$ est une fonction $\mathbb{N} \to \mathcal{T}_\mathbb{A} \to \mathbb{N}_\infty$).

Soit t_{E_m} l'arbre fini obtenu à partir de t en supprimant tous les noeuds après la frontière E_m.

L'arbre t' sera obtenu à partir de t_{E_m} par une itération appropriée de certains blocs sur toutes les branches.

Soit A l'ensemble des branches de t_{E_m}. A est un ensemble fini, partiellement ordonné par $<_{imb}$. On construit par récurrence des ensembles $(A_j)_{j\in\mathbb{N}}$ de la manière suivante : $A_0 = \emptyset$ et pour tout $j > 0$, A_j est l'ensemble des branches de t_{E_m} maximales pour $<_{imb}$ dans $A \setminus (\bigcup_{j'<j} A_{j'})$. Soit $p \in \mathbb{N}$ tel que $A_p \neq \emptyset$ et pour tout $j > p$, $A_j = \emptyset$. Alors $A_1, \ldots, A_p$ forme une partition de A, et pour tout $j \in [1, p]$, pour tout $\pi_1, \pi_2 \in A_j$, on a $\pi_1 \approx_{imb} \pi_2$. remarquons que pour tout $j \in [1, p]$ et tout $i \in [1, m]$, l'ensemble $\{e_i^\pi : \pi \in A_j\}$ est une frontière partielle de t_{E_m} (plus précisément, c'est l'intersection d'une frontière de t_{E_m} avec A_j).

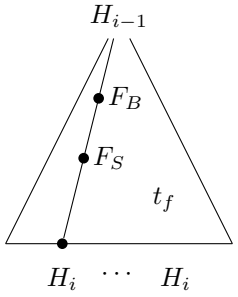
On peut maintenant adapater la technique utilisée par Rabin dans [Rab70], en commençant par se restreindre à A_1. Soit $N := |Q_{\mathcal{A}}|$. Dans la suite, les S-valeurs seront approchées en remplaçant toute valeur supérieure à N par la valeur $N+1$. On veut ici faire attention à ce que le pompage de [Rab70] ne réduise pas la S-valeur en-dessous de N. Une *valuation de* Γ_S est un élément de $[0, N+1]^{\Gamma_S}$, qui fixe une valeur pour chaque S-compteur, modulo l'approximation ci-dessus.

On peut voir une exécution sur un arbre t_e comme un étiquetage de t_e par $Q_{\mathcal{A}} \times [0, N+1]^{\Gamma_S}$: chaque noeud est étiqueté par l'état courant, ainsi que par la valeur courante des compteurs. Un élément (p, η) de $Q_{\mathcal{A}} \times [0, N+1]^{\Gamma_S}$ est donc appelé une *configuration* de l'automate.

On définit les ensembles de configurations $H_m, \ldots, H_0$ par récurrence descendante : on pose $H_m := Q_{\mathcal{A}} \times [0, N+1]^{\Gamma_S}$, et $(q, \eta) \in H_{i-1}$ si $(q, \eta) \in H_i$ et s'il existe un arbre fini t_f contenu dans t_{E_m}, ainsi qu'une exécution partielle R_f de $\mathcal{A}$ sur t_f partant d'une configuration (q, η) telle que

- toute branche de R_f est un bloc,
- sur toute feuille x de t_f, la configuration (q', η') décrite par R sur x est un élément de H_i.

Remarquons que certains noeuds de t_f peuvent avoir une arité 1 (ce sont les noeuds d'imbrication de t_{E_m}). Intuitivement, on veut être capable d'atteindre une frontière de configurations de H_i uniquement par des branches formant des blocs, en partant d'une configuration de H_{i-1} (qui est également dans H_i).



On a $H_0 \subseteq H_1 \subseteq \cdots \subseteq H_m$, et de plus l'exécution R sur les branches membres de A_1 est un témoin du fait que pour toute branche $\pi \in A_1$, pour tout $i \in [0, m]$,

la configuration (q, η) de $\mathcal{A}$ prise dans R en position e_i^π appartient toujours à H_i. De plus, il existe un arbre fini t_0 tel que $\mathcal{A}$ possède une exécution sur t_0 partant de $(q_0, 0)$ et terminant dans une configuration de H_0 sur toute feuille (t_0 est un sous-arbre de t_{E_m}).

L'approximation effectuée sur les S-valeurs est cohérente, puisque les seules changements de valeur possibles sont l'incrément et la remise à zéro. Ainsi les valeurs inférieures à N sont toujours exactes, et les valeurs supérieures à $N+1$ sont toujours envoyées sur la valeur $N+1$. De plus, aucun bloc de R ne peut checker une S-valeur inférieur à N, puisque $val_S(R) > N$. La valeur $N+1$ peut donc être interprétée comme « le check est autorisé sur cette valeur ».

On a $|H_m| = N\dot{(}N+2)^{|\Gamma_S|} \leq m$, donc il existe $i \in [0, m-1]$ tel que $H_i = H_{i+1}$. Cela signifie que pour toute configuration (p, η) de H_i, il existe un arbre fini t_f contenu dans t_{E_m}, et une exécution partielle de $\mathcal{A}$ sur t_f (contenue dans R) dont toute branche est un bloc, et dont toutes les feuilles sont étiquetées par une configuration de H_i. Ceci va nous permettre de créer par pompage un arbre infini, sur lequel $\mathcal{A}$ possède une exécution comportant une infinité de frontière de configurations de H_i. On veut conserver la B-valeur d'une telle exécution basse, ce qui explique la contrainte sur les B-compteurs dans la définition des blocs : pour tout B-coumpteur $\gamma \in \Gamma_B$, si un bloc incrément γ, alors il doit aussi le remettre à zéro. Concaténer plusieurs blocs de B-valeur bornée résulte en une augmentation bornée de la B-valeur : le pire cas possible est la concaténation d'un bloc de la forme $\texttt{ric}^{n_1}$ avec un autre de la forme $\texttt{ic}^{n_2}\texttt{r}$.

On définit $n_{\text{blocs}} := \max\{val_B(uv) : u, v \text{ dont des blocs de } R \text{ de } H_i \text{ à } H_i\}$, la B-valeur maximale obtenue en concaténant deux blocs de H_i. On peut remarquer que grâce à la définition des blocs, la concaténation de plus de deux blocs ne permet pas de dépasser cette B-valeur. On définit également n_{pref} comme la B-valeur de l'exécution partielle qui attient la première frontière de H_i, à partir de la racine en configuration $(q_0, 0)$, dans l'exécution R. Finalement, on pose $n_1 := n_{\text{blocs}} + n_{\text{pref}}$.

On construit ainsi un arbre infini (mais pas encore binaire complet : seules les branches de A_1 sont infinies) t_1, et une exécution partielle R_1 de $\mathcal{A}$ sur t_1, en utilisant R pour atteindre une frontière de H_i (les noeuds e_i^π pour $\pi \in A_1$), puis en répétant infiniment des arbres finis qui témoignent d'exécution partielles de positions de H_i vers des frontières de H_i.

La construction garantit que toute branche infinie π de R_1 est B-acceptante, avec $val_B(\pi) \leq n_1$, et S-acceptante avec $val_S(\pi) > N$, car aucune S-valeur inférieure à N n'est checkée pendant R.

L'arbre t_1 peut contenir une infinité de positions d'arité 1, ainsi qu'un nombre fini de feuilles (sur la frontière E_m). Ces positions d'arité 1 proviennent de duplications de noeuds d'imbrications entre les branches de A_1 et celles de $A_2, \ldots, A_p$. Soit A_2' l'ensemble des branches finies das t_1 qui sont copiés de préfixes de branches de A_2.

La condition d'imbrication garantit que le pompage des blocs de A_1 n'a pas endommagé les blocs des autres A_j : aucun bloc n'est coupé autrement que par un noeud d'imbrication. On peut donc définir les ensemble $H_m, \ldots, H_0$ comme précédemment, et pomper les branches de A_2' simultanément. On construit ainsi

un arbre t_2 satisfaisant les mêmes propriétés que t_1, mais possédant « plus » de branches infinies : les branches manquantes sont maintenant des préfixes de $A_3, \ldots, A_p$.

En répétant ce processus p fois, on obtient finalement un arbre binaire complet infini $t' := t_p$, ainsi qu'un nombre $n := n_1 + n_2 + \cdots + n_p$ (qui dépend seulement de l'ensembles fini des blocs définis par R sur t_{E_m}) tel qu'il existe une exécution R' de $\mathcal{A}$ sur t', témoignant à la fois $[\![\mathcal{A}]\!]_B(t') \leq n$ et $[\![\mathcal{A}]\!]^B_S(n)(t') > N$.

On va maintenant partir de t' pour construire une suite infinie d'arbres $(t'_d)_{d\in\mathbb{N}}$, tels que $d \in \mathbb{N}$, $[\![\mathcal{A}]\!]_B(t'_d) \leq n$, et $[\![\mathcal{A}]\!]^B_S(n)(t'_d) > N + d$.

Pour ce faire, considérer les S-actions effectuées par R' sur t'. Puisque $val_S(R') > N$, chaque action $\mathtt{cr}_\gamma$ pour $\gamma \in \Gamma_S$ dans R' doit être effectuée dans une configuration où la valeur de γ est au moins $N + 1$. Ceci signifie que le $\mathtt{cr}_\gamma$ est précédé par une séquence contenant $N + 1$ incréments de γ et aucun $\mathtt{r}_\gamma$ ou $\mathtt{cr}_\gamma$.

Par choix de N, une telle séquence doit contenir un chemin u_γ d'un certain $q \in Q_\mathcal{A}$ au même état q, commençant par $\mathtt{i}_\gamma$ et aucun reset pour γ.

On rappelle que les BS-actions de $\mathcal{A}$ sont hiérarchiques.Observons les différents comportements possibles d'un autre compteur $\gamma' \in \Gamma_B \cup \gamma_S$ sur le chemin u_γ.

- Si $\gamma' > \gamma$ alors le modifier provoque un $\mathtt{r}_\gamma$, donc γ' est laissé inchangé sur tout le chemin u_γ : la seule action autorisée pour γ' sur u_γ est ε.
- Si $\gamma' < \gamma$, alors u_γ commence par une action $\mathtt{r}_{\gamma'}$, induit par l'action i_γ.

Dans tous les cas, la répétition de u_γ plusieurs fois n'affecte pas la valeur des autres compteurs.

De plus, deux tels chemins u_γ et $v_{\gamma'}$ ne peuvent pas s'intersecter si $\gamma \neq \gamma'$. Soit t'_d l'arbre infini obtenu à partir de t' ainsi : pour tout $\gamma \in \Gamma_S$, pour toute position x étiquetée $\mathtt{cr}_\gamma$ dans R', le chemin u_γ relatif à cette position x est répété d fois. Pour tout $d \in \mathbb{N}$, l'exécution R' induit une exécution R'_d de $\mathcal{A}$ sur t'_d. Cette exécution est toujours B- et S-acceptante avec B-valeur au plus n, mais vérifie $val_S(R_d) > N + d$. En effet, d incréments supplémentaires ont été effectué avant chaque action $\mathtt{cr}$.

On peut maintenant tirer la conclusion voulue sur $\mathcal{U}$ et $\mathcal{U}'$. Soit α et $(\beta_i)_{i\in\mathbb{N}}$ les fonctions de corrections témoignant $\mathcal{A} \cong \mathcal{U} \times \mathcal{U}'$. En particulier $[\![\mathcal{A}]\!]_B \approx_\alpha [\![\mathcal{U} \times \mathcal{U}']\!]_B = [\![\mathcal{U}]\!]_B$ ($\mathcal{U}'$ ne joue aucun rôle dans la B-semantique de $\mathcal{U} \times \mathcal{U}'$). Ceci implique que pour tout $d \in \mathbb{N}$, $[\![\mathcal{U}]\!]_B(t'_d) \leq \alpha(n)$.

Or, on a aussi $[\![\mathcal{A}]\!]^B_S(n) \preccurlyeq_{\beta_n} [\![\mathcal{U} \times \mathcal{U}']\!]^B_S(\alpha(n))$. Soit $d \in \mathbb{N}$, on a en particulierr, $N + d \leq \beta_n([\![\mathcal{U} \times \mathcal{U}']\!]^B_S(\alpha(n))(t'_d))$.

Or, $[\![\mathcal{U} \times \mathcal{U}']\!]^B_S(\alpha(n))(t'_d)$ est défini comme le supremum de $val_S(R'')$, où R'' est une exécution S-acceptante de $\mathcal{U} \times \mathcal{U}'$ sur t'_d telle que $val_B(R'') \leq \alpha(n)$. Puisque l'on sait que $[\![\mathcal{U} \times \mathcal{U}']\!]_B(t'_d) \leq \alpha(n)$, il existe une exécution S-acceptante R'' de $\mathcal{U} \times \mathcal{U}'$ sur t'_d témoignant $\beta_n([\![\mathcal{U} \times \mathcal{U}']\!]^B_S(t'_d) \geq N + d$. Cela signifie que $\beta_n(val_S(R'')) \geq N + d$. mais R'' induit une exécution S-acceptante $R_{\mathcal{U}'}$ de $\mathcal{U}'$ sur t'_d avec $\beta_n(val_S(R_{\mathcal{U}'})) \geq N + d$.

Pour tout $d \in \mathbb{N}$, on obtient $\beta_n([\![\mathcal{U}']\!]_S(t'_d) \geq N+d$, donc $[\![\mathcal{U}']\!]_S$ n'est pas borné sur l'ensemble $\{t'_d : d \in \mathbb{N}\}$ alors que $[\![\mathcal{U}]\!]_B$ est borné sur le même ensemble.

Un piège de coût pour $\mathcal{A}$ implique donc bien $[\![\mathcal{U}']\!]_S \not\preccurlyeq [\![\mathcal{U}]\!]_B$. □

13.2.3 Construction du B-QWAA $\mathcal{B}$

On veut maintenant généraliser la construction de la Section 13.1.3 de manière à l'adapter aux automates de coût. Soient $\mathcal{U} = \langle Q_\mathcal{U}, \mathbb{A}, q_0^\mathcal{U}, F_B^\mathcal{U}, \Gamma_\mathcal{U}, \Delta_\mathcal{U}\rangle$ un B-NBA et $\mathcal{U}' = \langle Q_{\mathcal{U}'}, \mathbb{A}, q_0^{\mathcal{U}'}, F_S^{\mathcal{U}'}, \Gamma_{\mathcal{U}'}, \Delta_{\mathcal{U}'}\rangle$ un S-NBA tels que $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$. On a supposé ici que $\mathcal{U}$ et $\mathcal{U}'$ ont un seul état initial, il est facile de se ramener à ce cas. On veut construire un B-QWAA $\mathcal{B}$ équivalent à $\mathcal{U}$ et $\mathcal{U}'$.

Soit $\mathcal{H} = \langle Q_\mathcal{H}, \mathbb{C}, q_0^\mathcal{H}, \Gamma_B, \Gamma_S, F_B, F_S, \Delta_\mathcal{H}\rangle$ le hBS-T $\mathcal{H}(\Gamma_\mathcal{U}, \Gamma_{\mathcal{U}'})$ donné par le Théorème 12.4.1.

L'automate $\mathcal{B}$ possède les même compteurs que $\mathcal{U}$: son ensemble de compteur est $\Gamma_\mathcal{U}$.

On définit l'ensemble d'états de $\mathcal{B}$ par :

$$Q := Q_\mathcal{U} \times Q_{\mathcal{U}'} \times Q_\mathcal{H} \times (([1,m] \times \{\epsilon, \bot, \top\} \times \{\varepsilon, \mathtt{ic}, \mathtt{r}\})^{[0,|\Gamma_B|]} \cup \{q_\top\})$$

avec état initial $(q_0^\mathcal{U}, q_0^{\mathcal{U}'}, q_0^\mathcal{H}, \{(1, \epsilon, \varepsilon)\}^{[0,|\Gamma_B^\mathcal{H}|]})$.

En fait, les seuls états utiles seront ceux de $Q_{utile} \subset Q$: un élément $(q_1, q_2, q_3, (i, j, z))$ est dans Q_{utile} si pour tous $k, k' \in [0, |\Gamma_B^\mathcal{H}|]$, $k < k'$ implique $i(k) \leq i(k')$ et $j(k) \leq j(k')$, pour l'ordre $\epsilon < \bot < \top$. De plus, tout $(q_1, q_2, q_3, q_\top)$ est dans Q_{utile}.

Considérons $\mathcal{B}$ agissant sur un arbre $t \in \mathcal{T}_\mathbb{A}$. L'idée est qu'Eve choisit une exécution de $\mathcal{U}$ sur t et Adam choisit une exécution de $\mathcal{U}'$ sur t (techniquement, les joueurs choisissent les transitions à tour de rôle). De plus, Adam va choisir une unique branche, et contrôler le transducteur $\mathcal{H}$ sur les actions de $\mathcal{U}$ et $\mathcal{U}'$ sur cette branche. Les actions renvoyées par $\mathcal{B}$ sont celles renvoyées par $\mathcal{H}$.

Ainsi, une exécution de $\mathcal{B}$ sur t décrit une branche π de t, une exécution partielle u de $\mathcal{U}$ et u' de $\mathcal{U}'$ sur cette branche, ainsi qu'une exécution de $\mathcal{H}$ sur (u, u').

Il est cohérent de donner à Adam le contrôle de $\mathcal{H}$ puisque les objectifs de $\mathcal{U}'$ et $\mathcal{H}$ sont analogues : maximiser la S-valeur, sans contrôle significatif sur la B-valeur.

Une portion π d'exécution de $\mathcal{B}$ est appelée *bloc* si les conditions suivantes sont vérifiées :

– π contient un état de $F_B^\mathcal{U}$ et un état de $F_S^{\mathcal{U}'}$,

– pour tout compteur $\gamma \in \Gamma_B^\mathcal{H}$, si π contient un $\mathtt{ic}_\gamma$, alors π contient un $\mathtt{r}_\gamma$.

Soit $K = |\Gamma_B^\mathcal{H}|$, on note $\Gamma_B^\mathcal{H} = \{\gamma_1, \ldots, \gamma_K\}$. Si $k \in [1, k]$, on notera $\mathtt{ic}_k, \mathtt{r}_k, \varepsilon_k$ les actions effectuées sur γ_k.

Si $A \subseteq \mathbb{C}_B$ est un ensemble d'actions atomiques, on appellera A-séquence pour γ une portion d'exécution qui n'effectue que des actions de A sur γ.

On appelle *bloc de niveau* k un bloc contenu dans une $\{\varepsilon, \mathtt{r}\}$-séquence de γ_{k+1} (un bloc de niveau K est juste un bloc normal).

On donne maintenant une interprétation intuitive de l'ensemble Q. Etant donné un état $(q_{\mathcal{U}}, q_{\mathcal{U}'}, q_{\mathcal{H}}, (i,j,z)) \in Q$, et $k \in [0,K]$, $i(k) \in [1,m]$ est le numéro du bloc courant de niveau k. La composante $j(k)$ est ϵ si aucun état de $F_S^{\mathcal{U}'}$ n'a été vu dans le bloc, $\bot$ si $F_S^{\mathcal{U}'}$ a été vu mais pas pas encore suivi de $F_B^{\mathcal{U}}$, et $\top$ si $F_B^{\mathcal{U}}$ a été vu après $F_S^{\mathcal{U}'}$. Finalement, $z(k) = \varepsilon$ si aucun incrément ou reset pour γ_k n'a été vu dans le bloc courant, `ic` s'il y a eu un incrément sans reset, et `r` si un $\mathtt{r}_k$ a été vu dans le bloc.

L'état $(q_{\mathcal{U}}, q_{\mathcal{U}'}, q_{\mathcal{H}}, (i,j,z))$ est acceptant si la dernière composante vérifie pour tout k, $j(k) \neq \bot$, ou si c'est $q_\top$.

En accord avec la définition des blocs, un bloc de niveau k peut être fermé si $j(k) = \top$ et $z(k) \in \{\varepsilon, \mathtt{r}\}$.

Pour tout $k \in [0,K]$, l'automate commence en $i(k) = 1$.

Une fois que les transitions de $\mathcal{U}$ et $\mathcal{U}'$ sont fixées, on définit comment $\mathcal{B}$ met à jour ses composantes i, j, z de manière déterministe.

L'automate $\mathcal{B}$ va donc compter les blocs, de manière à chercher à construire un piège de coût. Les actions effectuées par $\mathcal{B}$ sont les B-actions effectuées par $\mathcal{U}$.

On dira qu'on *redémarrre* un niveau k pour signifier que pour tout $k' \leq k$, $i(k')$ est remis à 1, $j(k)$ à ϵ, et $z(k)$ à `r`. On rappelle que les actions des compteurs de $\mathcal{B}$ proviennent du hBS-T $\mathcal{H}$, et sont donc hiérarchiques.

(1) Si un état de $F_S^{\mathcal{U}'}$ est vu, tous les $j(k) = \epsilon$ sont mis à $\bot$.
(2) Si un état de $F_B^{\mathcal{U}}$ est vu, tous les $j(k) = \bot$ sont mis à $\top$.
(3) Si une action $\mathtt{r}_k$ est vue (avec $\varepsilon_{k'}$ pour $k' > k$ et $\mathtt{r}_{k'}$ pour $k' \leq k$), alors $z(k)$ est mis à `r` pour tout $k' \leq k$.
(4) Si une action $\mathtt{ic}_k$ est vue (avec $\varepsilon_{k'}$ pour $k' > k$ et $\mathtt{r}_{k'}$ pour $k' < k$), alors l'automate redémarre le niveau $k-1$. De plus, pour tout $k' \geq k$ tel que $z(k') = \varepsilon$, $z(k')$ est mis à `ic`.
(5) Après les mises à jour (1) à (4), s'il existe k tel que $j(k) = \top$ et $z(k) \in \{\varepsilon, \mathtt{r}\}$, alors $i(k)$ est incrémenté, $j(k)$ est mis à ϵ et $z(k)$ à ε.
(6) Si un certain $i(k)$ doit incrémenter d'après (5) mais vaut déjà m, alors on passe dans la composante $q_\top$, et on continue à simuler $\mathcal{U}$ et $\mathcal{U}'$. On ne peut pas sortir de la composante $q_\top$ donc l'exécution est acceptante.

Il est facile de vérifier qu'en respectant ces règles, les seuls états accessibles par $\mathcal{B}$ sont dans Q_{utile}.

Cette description informelle de $\mathcal{B}$ devrait permettre de reconstituer la fonction de transition $\delta : Q \times \mathbb{A} \to \mathcal{B}^+(\{0,1\} \times \mathbb{C}_B^{\Gamma_{\mathcal{U}}} \times Q)$, dont la définition formelle est très lourde.

13.2.4 Quasi-faiblesse de $\mathcal{B}$

On montre dans cette section que $\mathcal{B}$ est bien un B-QWAA.

Soit π une n-exécution de $\mathcal{B}$ (c'est-à-dire de B-valeur au plus n). On définit une configuration $C = (i_C, v_C, j_C, z_C)$ de $\mathcal{B}$ comme état un élément de $E_n = ([1,m] \times [0,n] \times \{\epsilon, \bot, \top\} \times \{\varepsilon, \mathtt{ic}, \mathtt{r}\})^{K+1} \cup \{q_\top\}$.

Le nouvel élément v_C stocke la valeur du compteur k dans $v_C(k)$ pour tout $k \in [1,K]$. Puisque le niveau 0 ne correspond à aucun compteur, on fixe toujours

la valeur de $v_C(0)$ à 0. Pour les autres composantes, l'ensemble $[0, n]$ suffit car π est une n-exécution, donc les B-compteurs ne dépassent jamais la valeur n.

On analyse maintenant l'évolution de la configuration de $\mathcal{B}$ le long du chemin π, en accord avec les règles (1) à (6).

Une *alternation* est un changement entre état acceptant et non-acceptant de l'automate.

On va montrer que si une portion de π commence et finit dans la même configuration C, alors elle ne comporte pas d'alternation. Une telle portion est appelée un cycle.

Soit $\mathcal{C}$ un cycle commençant et finissant dans une configuration C. Si aucun $i(k)$ n'est modifié durant $\mathcal{C}$, alors les autres éléments ne peuvent qu'augmenter pour les ordres $\epsilon < \bot < \top$ et $\varepsilon < \mathtt{ic} < \mathtt{r}$, d'après les règles (1) à (6). Le cycle $\mathcal{C}$ contient donc une seule configuration, et par conséquent ne contient pas d'alternation.

Autrement, soit k maximal tel que $i(k)$ est modifié dans $\mathcal{C}$. Pour pouvoir retourner à C, le cycle doit contenir un reset pour $i(k)$. Remarquons que la règle (4) est la seule à pouvoir effectuer un reset de $i(k)$, et cela arrive quand un $\mathtt{ic}_{k'}$ est vu, avec $k' > k$. Soit k' tel que $\mathtt{ic}_{k'}$ est vu sur $\mathcal{C}$. D'après les règles (1) et (2), $j(k')$ ne peut qu'incrémenter durant $\mathcal{C}$, puisque le niveau k' n'est pas redémarré (par maximalité de k). Puisque $\mathcal{C}$ est un cycle, $j(k')$ est constant durant $\mathcal{C}$.

On considère les différents cas possibles pour la valeur constante de $j(k')$. Si c'est $\bot$, alors $\mathcal{C}$ est entièrement non-acceptant, et ne contient donc pas d'alternation.

On sait que $v(k')$ est incrémenté à un moment par l'action $\mathtt{ic}_{k'}$, donc le compteur k' doit être remis à zéro durant $\mathcal{C}$. Si la valeur de $j(k')$ est $\top$, cela signifie qu'on a $z(k') = \mathtt{r}$ et $j(k') = \top$ à la fin du cycle. Or la règle (5) force alors un incrément de $i(k')$, ce qui contredit la maximalité dans le choix de k.

Le seul cas non encore traité est celui où $j(k') = \epsilon$ durant tout le cycle $\mathcal{C}$. Par définition de Q_{utile}, pour tout $k'' \leq k'$, $j(k'') = \epsilon$ sur $\mathcal{C}$. De plus, pour tout $k'' > k'$, $j(k'')$ ne peut qu'incrémenter sur $\mathcal{C}$ (par choix de k'), et ne peut donc pas changer durant le cycle $\mathcal{C}$. Les états acceptants étant déterminés seulement par la composante j, le cycle $\mathcal{C}$ est aussi sans alternation dans ce cas.

Tout cycle est sans alternation, donc le nombre d'alternations dans une n-exécution est borné par le nombre d'éléments de E_n, qui est inférieur à $\alpha(n) = ((m+1)(n+1)9)^{K+1} + 1$. Puisque K et m sont fixés par $\mathcal{U}$ et $\mathcal{U}'$ et ne dépendent pas de l'arbre d'entrée t, $\mathcal{B}$ est bien un B-automate quasi-faible.

De plus, on n' pas utilisé la supposition selon laquelle $\mathcal{U}$ et $\mathcal{U}'$ sont équivalents jusqu'à maintenant. Cela signifie que $\mathcal{B}$ ainsi construit sera toujours un B-QWAA, sans condition les sémantiques de $\mathcal{U}$ et $\mathcal{U}'$. Ce fait sera utilisé dans la Section 13.3.3.

13.2.5 Correction du B-QWAA $\mathcal{B}$

Preuve de $[\![\mathcal{B}]\!]_B \preccurlyeq [\![\mathcal{U}]\!]_B$

On va en fait montrer $[\![\mathcal{B}]\!]_B \leq [\![\mathcal{U}]\!]_B$, et ce sans utiliser l'hypothèse $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$.

Soit $t \in \mathcal{T}_{\mathbb{A}}$ un arbre infini et $M \in \mathbb{N}$tel que $[\![\mathcal{U}]\!]_B(t) = M < \infty$. On veut montrer $[\![\mathcal{B}]\!]_B(t) \leq M$. Soit R une exécution B-acceptante de $\mathcal{U}$ sur t, de valeur M.

Soit σ la stratégie pour Eve dans $(\mathcal{B}, t)$ consistant à jouer l'exécution R. Puisque les B-actions de $\mathcal{B}$ sont directement copiées de $\mathcal{U}$, on a $val_B(\sigma) = val_B(R) = M$ (en ignorant la condition d'acceptation).

Il reste à montrer que σ est gagnante, c'est-à-dire que toute exécution de $\mathcal{B}$ compatible avec σ est acceptante. Supposons pour la contradiction qu'il existe une exécution π de $\mathcal{B}$ compatible avec σ, qui se stabilise dans des états non-acceptants.

Par définitions des états non-acceptants de $\mathcal{B}$, il existe $k \in [0, m]$ avec $j(k) = \bot$ dans tout état de π à partir d'un certain point. Soit k maximal tel que $j(k) = \bot$ infiniment souvent dans π.

On montre qu'à partir d'un certain point, $j(k)$ vaut toujours $\bot$. Si k est le compteur maximal, alors le niveau k n'est jamais redémarré, et $j(k)$ se stabilise bien en $\bot$. Sinon, supposons que $j(k)$ ne se stabilise pas en $\bot$. Il y a donc une infinité de mises à jour de $j(k)$ de ϵ à $\bot$, et une infinité de redémarrages du niveau k. Soit $k' > k$. Par choix de k, $j(k')$ doit se stabiliser sur ϵ ou $\top$ (sinon, $j(k')$ vaut $\bot$ infiniment souvent). Si $j(k')$ se stabilise en ϵ, toute mise à jour de $j(k)$ de ϵ à $\bot$ est effectuée simultanément sur $j(k')$, c'est donc absurde. Si $j(k')$ se stabilise en $\bot$, tout redémarrage du niveau k est soit accompagné d'un redémarrage pour le niveau k', soit causé par une action $\mathtt{r}$ sur le compteur k'. Dans les deux cas on a une absurdité, puisque $j(k') = \top$ et $z(k') = \mathtt{r}$ provoque un incrément de i et met à jour $j(k')$ à ϵ (ou alors m est atteint et l'automate passe dans la composante $q_\top$).

Ceci montre que $j(k')$ se stabilise en $\bot$ dans π.

Or, R est acceptante donc il y a une infinité d'états acceptants $F_B^{\mathcal{U}}$ de $\mathcal{U}$ sur π. Par l'opération (2) de $\mathcal{B}$, $j(k)$ ne peut donc pas se stabiliser en $\bot$ sur π.

On a donc atteint une absurdité, ce qui montre bien que σ est acceptante, et $[\![\mathcal{B}]\!](t) \leq [\![\mathcal{U}]\!](t)$.

C'est vrai pour tout t donc $[\![\mathcal{B}]\!]_B \leq [\![\mathcal{U}]\!]_B$. De plus, on n'a pas utilisé l'hypothèse $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$, cette inégalité est donc vraie pour tous $\mathcal{U}$ et $\mathcal{U}'$.

Preuve de $[\![\mathcal{U}]\!]_B \preccurlyeq [\![\mathcal{B}]\!]_B$

On suppose pour la contradiction que $[\![\mathcal{U}]\!]_B \not\preccurlyeq [\![\mathcal{B}]\!]_B$, c'est-à-dire qu'il existe une suite d'arbres infinis $(t_n)_{n\in\mathbb{N}}$ telle que $\{[\![\mathcal{B}]\!]_B(t_n), n \in \mathbb{N}\}$ est borné par un certain $M \in \mathbb{N}$, mais $\{[\![\mathcal{U}]\!]_B(t_n), n \in \mathbb{N}\}$ n'est pas borné. On rappelle que puisque $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$, l'ensemble $\{[\![\mathcal{U}']\!]_S(t_n), n \in \mathbb{N}\}$ est également non borné. On va obtenir une contradiction en construisant un piège de coût pour $\mathcal{A} \cong \mathcal{U} \times \mathcal{U}'$.

Soit $\mathcal{A}$ le BS-NBA $\mathcal{H} \circ (\mathcal{U} \times \mathcal{U}')$. C'est-à-dire,

– $Q_{\mathcal{A}} := Q_{\mathcal{U}} \times Q_{\mathcal{U}'} \times Q_{\mathcal{H}}$ et $q_0^{\mathcal{A}} := (q_0^{\mathcal{U}}, q_0^{\mathcal{U}'}, q_0^{\mathcal{H}})$,
– $\Gamma_B^{\mathcal{A}} := \Gamma_B^{\mathcal{H}}$ et $\Gamma_S^{\mathcal{A}} := \Gamma_S^{\mathcal{H}}$,
– $F_B^{\mathcal{A}} := F_B^{\mathcal{U}}$ et $F_S^{\mathcal{A}} := F_S^{\mathcal{U}'}$,
– $\Delta_{\mathcal{A}} = \Delta_{\mathcal{H}} \circ (\Delta_{\mathcal{U}} \times \Delta_{\mathcal{U}'})$.

Rappelons que par le Théorème 12.5.1, $\mathcal{A}$ est un hBS-NBA tel que $\mathcal{A} \approxeq \mathcal{U} \times \mathcal{U}'$.

On rappelle que $\mathcal{H}$ est un BS-T fidèle et hd. Soit $\mathcal{G}$ le BS-NBA sur $\mathbb{C}$ qui copie simplement l'action lue en entrée. Alors par le Théorème 12.4.1, $\mathcal{G} \approxeq \mathcal{H} \circ \mathcal{G} = \mathcal{H}$.

Soit $\delta = (\delta_n)_{n\in\mathbb{N}}$ une famille de stratégies monotones témoignant du déterminisme en histoire de $\mathcal{H}$. Si $\mathcal{H}^\delta$ est le BS-T $\mathcal{H}$ restreint à ces stratégies, alors par le Théorème 12.4.1, $\mathcal{H}^\delta \circ \mathcal{G} \approxeq \mathcal{G}$. Soit α et $(\beta_m)_{m\in\mathbb{N}}$ des fonctions de correction témoins de cette dernière équivalence.

Soit $N := \beta_M(|Q_{\mathcal{A}}| + 1)$, et $n \in \mathbb{N}$ tel que $[\![\mathcal{U}']\!]_S(t_n) > N$. Soit R' une exécution acceptante de $\mathcal{U}'$ sur $t := t_n$ telle que $val_S(R') > N$.

Soit σ une M-stratégie gagnante pour Evedans le jeu $(\mathcal{B}, t)$ (donc avec $val_B(\sigma) \leq M$). On considère en particulier l'ensemble P des exécutions de $\mathcal{B}$ compatibles avec σ où Adam joue l'exécution R'et utilise la stratégie $\delta_{|Q_{\mathcal{A}}|+1}$ pour diriger $\mathcal{H}$, c'est-à-dire que l'on fixe toutes les décisions d'Adam sauf le choix d'une branche de t. La seule marge de manoeuvre dans P étant le choix d'une branche, P décrit un arbre d'exécutions possibles, qui ne dépendent pas à l'avance de la branche choisie.

Pour obtenir $val_B(\sigma) \leq M$, Eve doit jouer une exécution R de $\mathcal{U}$ telle que $val_B(R) \leq M$ sur toute branche de P. Soit π une branche de l'arbre P.

On commence par montrer que la stratégie $\delta_{|Q_{\mathcal{A}}|+1}$ permet à $\mathcal{H}$ d'accepter le mot $v = (out_B(R, \pi), out_S(R', \pi))$ où $out_B(R, \pi)$ (resp. $out_S(R', \pi)$) sont les mots infinis décrivant les actions de R (resp. R') sur la branche π. Puisque $\mathcal{H}^\delta \approxeq_\alpha \mathcal{G}$, il existe une exécution $R_{\mathcal{H}}$ de $\mathcal{H}$ sur v dirigée par une stratégie δ_p avec $val_B(R_{\mathcal{H}}) \leq \alpha(M)$ et $\beta_M(val_S(R_{\mathcal{H}})) \geq N = \beta_M(|Q_{\mathcal{A}}|+1)$. Par conséquent, soit $p = val_S(R_{\mathcal{H}}) \geq |Q_{\mathcal{A}}| + 1$, soit $val_S(R_{\mathcal{H}}) = \infty$. Dans les deux cas, par monotonicité de δ, tl'exécution de $\mathcal{H}$ sur v dirigée par $\delta_{|Q_{\mathcal{A}}|+1}$ est acceptante.

Cela signifie que toute branche de P représente une exécution acceptante de $\mathcal{H}$ surle mot v correspondant, et renvoie de plus des BS-actions hiérarchiques de S-valeur au moins N et B-valeur au plus M.

On montre maintenant que toute exécution de P doit contenir m blocs de coûts correctement agencés, prouvant ainsi que P décrit un piège de coût pour $\mathcal{A}$.

Soit π une branche de P.

Si l'automate $\mathcal{B}$ atteint l'état $q_\top$ sur π, alors pour un certain niveau k, m blocs de niveau k ont été vus.

Sinon, soit k minimal tel que $i(k)$ ne change qu'un nombre fini de fois dans π, et se stabilise donc sur une certaine valeur. (c'est toujours le cas pour $i(K)$, qui ne peut qu'augmenter).

On sait que $j(k)$ se stabilise en ϵ ou $\top$, car π est acceptante. Mais ϵ est impossible, car R' étant acceptant, il y a une infinité d'états de $F_S^{\mathcal{U}'}$ sur π, qui changeraient $j(k) = \epsilon$ en $\bot$ par la règle (1). On peut donc affirmer que $j(k)$ se stabilise en $\top$. Il y a donc un nombre fini de remises à zéro du compteur k, sans

quoi $i(k)$ serait incrémenté à nouveau par la règle (5). Puisque π a une B-valeur bornée, on peut conclure que le nombre d'incréments de tout compteur $k' \geq k$ sur π est également fini. Mais $i(k-1)$ est supposé changé infiniment souvent, ce qui ne peut arriver qu'en incrémentant infiniment souvent un compteur $k' \geq k$, par la règle (4). C'est absurde, donc π se stabilise en $q_\top$, et contient donc m blocs consécutifs.

On construit maintenant un piège de coût pour le hBS-NBA $\mathcal{A} \cong \mathcal{U} \times \mathcal{U}'$. Soit $R_\mathcal{A}$ l'exécution de $\mathcal{A}$ correspondant à l'arbre P défini plus tôt, en ignorant simplement les composantes de $\mathcal{B}$ qui n'apparaissent pas dans $\mathcal{A}$ (ce sont celles qui permettent de compter les blocs). On a $val_B(R_\mathcal{A}) \leq M$ et $val_S(R_\mathcal{A}) > |Q_\mathcal{A}|$.

On a montré que sur toute branche π de $R_\mathcal{A}$, l'état $q_\top$ est atteint, et donc m blocs consécutifs de même niveau sont présents sur cette branche.

Pour toute branche π de $R_\mathcal{A}$, pour tout $i \in [0, m]$, on choisit pour e_i^π la dernière position π où un compteur $i(k)$ a valeur i.

Par définition ddes transitions de $\mathcal{B}$ (règles (1) à (6)), la portion d'exécution $[e_i^\pi, e_{i+1}^\pi)$ est toujours un bloc.

Il reste à montrer que la condition d'imbrication dans la définition des pièges de coût est respectée. Soit π_1, π_2 deux branches de $R_\mathcal{A}$, se séparant sur la position y, et soit $x < y$ la première position où leurs blocs diffèrent : $e_i^{\pi_1} = x$ mais $e_i^{\pi_2} > x$.

Par définition des e_i^π, x est la dernière position sur π_1 où un compteur $i(k)$ a valeur i. Supposons $e_i^{\pi_2} \leq y$, alors $e_i^{\pi_2}$ se trouve également sur π_1, or l'un des compteurs $i(k)$ a valeur i à cet endroit, ce qui est absurde puisque $e_i^{\pi_2} > x$. On peut en conclure que $e_i^{\pi_2} > y$: la condition d'imbrication est vérifiée.

On a achevé la preuve que $R_\mathcal{A}$ témoigne bien de l'existence d'un piège de coût pour $\mathcal{A}$. Par la proposition 13.2.2, ceci implique $[\![\mathcal{U}']\!]_S \not\preccurlyeq [\![\mathcal{U}]\!]_B$, ce qui est absurde ici.

La supposition initiale est donc fausse, on en conclut $[\![\mathcal{U}]\!]_B \preccurlyeq [\![\mathcal{B}]\!]_B$

13.3 Application à un problème de décision

13.3.1 Définitions

Soit $i \leq j$ deux entiers, avec $i \in \{0, 1\}$. Un automate à parité de rang $[i, j]$ est un automate muni d'une fonction $col : Q \to [i, j]$ qui associe une couleur de $[i, j]$ à chaque état. Cette fonction est utilisée dans la condition d'acceptation : une suite d'états $p_0 p_1 \ldots$ est acceptante si la plus grande couleur apparaissant infiniment souvent dans la suite est paire.

Cette condition d'acceptation permet de définir les automates à parité non-déterministes, alternant, ainsi que les B- et S-automates à parité.

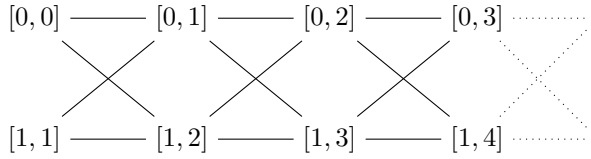
En particulier, un automate de Büchi est un automate à parité de rang $[1, 2]$: les états de Büchi possèdent la couleur 2, et les autres états la couleur 1.

On s'intéressera principalement aux automates non-déterministes dans cette section. Excepté pour les automates faibles alternant, tous les automates considérés seront non-déterministes. Dans la suite, on dira qu'un langage d'arbres infinis

est $[i,j]$-reconnaissable (resp. Büchi) s'il peut être reconnu par un automate de rang $[i,j]$ (resp. de Büchi) non-déterministe.

13.3.2 Etat de l'art

La hiérarchie de langages induite par les rangs $[i,j]$ d'automates non déterministes, dite hiérarchie non-déterministe de Mostowski, est stricte [Niw86].



La question de la décidabilité de cette hiérarchie est un problème ouvert majeur de la théorie des langages réguliers d'arbres infinis. Ce problème peut se formuler ainsi : étant donné un langage régulier L d'arbres infinis et un rang $[i,j]$, peut-on décider si L est $[i,j]$-reconnaissable ?

Plusieurs progrès ont été faits sur cette question, en particulier Walukiewicz et Niwinski [NW05] ont montré que ce problème est décidable lorsque L est donné via un automate déterministe. Cependant, comme il n'est pas toujours possible de déterminiser un automate d'arbres infinis, cela ne suffit pas à répondre à la question générale.

Plus récemment, Colcombet et Löding [CL08] ont réduit ce problème à celui de l'existence d'une borne pour les B-automates à parité :

Théorème 13.3.1 ([CL08]) *Soit L un langage régulier d'arbres infinis, et $[i,j]$ un rang. On peut construire un B-automate $\mathcal{U}$ non-déterministe de rang $[i,j]$ tel que $[\![\mathcal{U}]\!] \approx \chi_L$ si et seulement si L est $[i,j]$-reconnaissable. Ceci permet de construire un B-automate $\mathcal{V}$ non-déterministe de rang $[i,j]$ tel que $[\![\mathcal{V}]\!] \approx 0$ si et seulement si L est $[i,j]$-reconnaissable.*

Décider si un B-automate à parité est borné permettrait donc de décider l'indice de Mostowski d'un langage régulier. Malheureusement, ce problème est toujours ouvert, même pour le cas Büchi, correspondant au rang $[1,2]$.

On peut cependant utiliser le nouvel outil des B-QWAA pour apporter ici une contribution : si L est donné par un automate de Büchi, alors on peut décider si son complément est également Büchi. En d'autres termes, on peut décider si un langage Büchi est faible, par le Théorème de caractérisation de Rabin [Rab70]. Pour ce faire, on va utiliser la construction de [CL08] pour le rang $[1,2]$, ainsi que la construction de la Section 13.2.

13.3.3 Décider si un langage Büchi est faible.

Soit $\mathcal{A}$ un automate de Büchi non-déterministe reconnaissant un langage L. Soit $\mathcal{U}'$ l'automate $\mathcal{A}$ vu comme un S-NBA sans compteur. On a alors $[\![\mathcal{U}']\!]_S = \chi_{\overline{L}}$.

Soit $\mathcal{A}'$ un automate non-déterministe, complément de $\mathcal{A}$, donc reconnaissant $\overline{L}$. On sait construire $\mathcal{A}'$, mais ce sera a priori un automate à parité, et non un automate de Büchi. On peut maintenant utiliser le Théorème 13.3.1 sur $\mathcal{A}'$ pour construire un B-NBA $\mathcal{U}$ tel que $[\![\mathcal{U}]\!]_B \approx \chi_{\overline{L}}$ si et seulement si $\overline{L}$ est Büchi.

Appliquons maintenant l'algorithme de la section 13.2 sur le B-NBA $\mathcal{U}$ et le S-NBA $\mathcal{U}'$, afin de construire un B-QWAA $\mathcal{B}$. On rappelle que si $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$, alors $[\![\mathcal{B}]\!] \approx [\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S = \chi_{\overline{L}}$.

Lemme 13.3.2 *$[\![\mathcal{B}]\!]_B \approx \chi_{\overline{L}}$ si et seulement si L est Büchi.*

Démonstration Si L est Büchi alors par définition de $\mathcal{U}$, $[\![\mathcal{U}]\!]_B \approx \chi_{\overline{L}}$. On a alors $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$, et donc $[\![\mathcal{B}]\!] \approx \chi_{\overline{L}}$ par construction de $\mathcal{B}$.

Réciproquement, supposons $[\![\mathcal{B}]\!] \approx \chi_{\overline{L}}$. Par le Théorème de Simulation 11.3.4, on peut construire un B-NBA $\mathcal{U}'' = \langle Q, \mathbb{A}, In, F, \Gamma, \Delta \rangle$ tel que $[\![\mathcal{U}'']\!] \approx [\![\mathcal{B}]\!] \approx \chi_{\overline{L}}$. Cela signifie qu'il existe $M \in \mathbb{N}$ tel que $[\![\mathcal{U}'']\!](t) \leq M$ si $t \in \overline{L}$, et $[\![\mathcal{U}'']\!](t) = \infty$ si $t \in L$.

Soit $\mathcal{A}'' = \langle Q \times [0, M]^\Gamma, \mathbb{A}, In \times \{0\}^\Gamma, F \times [0, M]^\Gamma, \Delta' \rangle$ l'automate de Büchi obtenu à partir de $\mathcal{U}'$, en comptant exactement les valeurs des compteurs jusqu'à M, et en refusant l'entrée si l'un des compteurs dépasse M (par absence de transition par exemple).

Si $c \in [0, M]^\Gamma$ est un vecteur de valeurs pour les compteurs de Γ, et $\nu \in \mathbb{C}_B^\Gamma$ est une action sur ces compteurs, on note $\nu(c)$ le vecteur résultant de l'action ν sur les valeurs c.

Formellement, les transitions de $\mathcal{A}''$ sont définies par

$$\Delta' = \left\{ ((p, c), a, (q, \nu(c))) : (p, a, \nu, q) \in \Delta \text{ et } c, \nu(c) \in [0, M]^\Gamma \right\}.$$

Il est alors facile de montrer que $\mathcal{A}''$ est un automate de Büchi reconnaissant $\overline{L}$. On en conclut que $\overline{L}$ est bien Büchi.

□

Lemme 13.3.3 *Si $\mathcal{B}$ est un B-QWAA et L est un langage régulier, on peut décider si $[\![\mathcal{B}]\!]_B \approx \chi_{\overline{L}}$.*

Démonstration Soit $\mathcal{A}$ un automate non-déterministe à parité pour L, et $\mathcal{A}'$ un automate non-déterministe à parité pour $\overline{L}$. Alors $[\![\mathcal{A}]\!]_S = \chi_{\overline{L}} = [\![\mathcal{A}']\!]_B$. On a donc $[\![\mathcal{B}]\!]_B \approx \chi_{\overline{L}}$ si et seulement si $[\![\mathcal{A}]\!]_S \preccurlyeq [\![\mathcal{B}]\!]_B \preccurlyeq [\![\mathcal{A}']\!]_B$.

Or, d'après le Théorème 11.3.4, il existe un B-NBA $\mathcal{U}$ et un S-NBA $\mathcal{U}'$ tels que $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{B}]\!]_B \approx [\![\mathcal{U}']\!]_S$. On est donc ramené à décider $[\![\mathcal{A}]\!]_S \preccurlyeq [\![\mathcal{U}]\!]_B$ et $[\![\mathcal{U}']\!]_S \preccurlyeq [\![\mathcal{A}']\!]_B$.

Ces deux propriétés sont décidables, car elles se ramènent à trouver une exécution vérifiant certaines propriétés dans les automates produits correspondants. De manière générale, toute propriété de type $f \preccurlyeq g$ est décidable si f est donnée par un S-automate non-déterministe et g est donnée par un B-automate non-deterministe. Des détails à ce sujet peuvent être trouvés dans [Col11].

Ceci achève la preuve du lemme. □

En réunissant ces deux lemmes, on obtient le théorème suivant :

Théorème 13.3.4 *Etant donné un automate de Büchi non-déterministe reconnaissant un langage L, la question de savoir si L est faible (c'est-à-dire $\overline{L}$ Büchi) est décidable.*

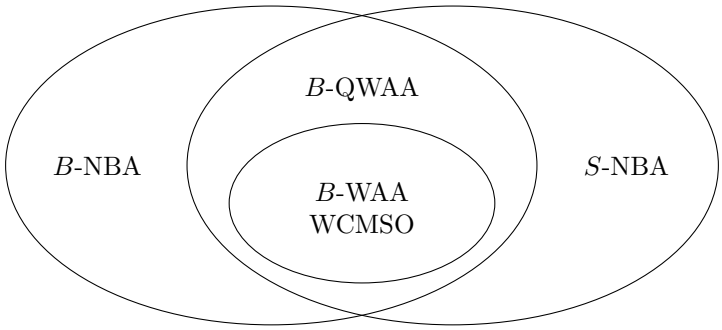
De plus, en utilisant le fait que l'on peut obtenir un automate non-déterministe Büchi à partir d'un alternant Büchi [MS95], et que le complémentaire d'un alternant co-Büchi est un alternant Büchi, on obtient le corollaire suivant :

Corollaire 13.3.5 *Etant donné un automate alternant Büchi ou co-Büchi reconnaissant un langage L, on peut décider si L est faible.*

On a cependant utilisé une construction complexe, celle de la Section 13.2, conçue pour être appliquée sur un B-NBA et un S-NBA. Ici, l'un des deux automates ne possède en fait pas de compteurs. On peut tirer parti de ce fait pour construire de manière « ad hoc » un B-QWAA possédant les propriétés voulues, en réutilisant les idées de [CL08] et de [KV99]. On ne détaillera pas cette construction ici.

13.4 Conclusion

Les résultats de ce chapitre sont résumés sur le schéma suivant. Toutes les inclusions représentées ici sont strictes. Celles que l'on n'a pas montrées se déduisent du cas des langages classiques : la classe des langages Büchi n'est pas close par complément [Rab70].



La classe des fonctions de coût quasi-faible présente une nouveauté par rapport au cadre des langages d'arbres infinis. En effet, une telle classe intermédiaire entre les langages faibles et les langages Büchi n'existe pas dans le cadre booléen, et constitue donc un saut qualitatif spécifique à la théorie des fonctions de coût. Il est intéressant de constater que malgré son caractère spécifique aux fonctions de coût, cette nouvelle classe nous permet d'obtenir un résultat de décidabilité sur les langages classiques : étant donné un automate de Büchi, on peut décider si le langage qu'il reconnaît est faible. Ce résultat est une contribution de la

thèse, et sa démonstration semble difficile à reformuler de manière à ne pas faire appel à la théorie des fonctions de coût. On peut donc espérer qu'en tant que nouvel outil, cette théorie permettra de répondre à des questions encore ouvertes.

On rappelle cependant que dans le cas des arbres infinis, beaucoup de travail reste à faire pour étayer la théorie des fonctions de coût. En particulier, la notion de fonction de coût régulière n'a pas encore de sens dans ce cadre, puisque l'équivalence entre B- et S-automates non-déterministes à parité n'a pas été montrée. La logique CMSO sur arbres infinis n'a donc pour l'instant pas d'équivalent en termes d'automates, et sa décidabilité est un problème ouvert.

Chapitre 14

Conclusion et perspectives

On a montré de nombreux résultats sur le pouvoir expressif de divers formalismes de reconnaissance de fonctions de coût. Ces résultats peuvent se classer en trois catégories. Premièrement, il y a les généralisations des résultats sur les langages classiques, qui montrent une certaine robustesse de la classe des fonctions de coût. Les résultats de ce type sont ceux obtenus sur mots finis et infinis, caractérisant les fonctions de coût définissables par CLTL et Prompt-LTL, ainsi que le résultat d'équivalence entre CMSO et WCMSO. Il y a ensuite des résultats spécifiques aux fonctions de coût : par exemple ceux obtenus durant l'étude de la classe temporelle, ou les résultats de formes normales d'automate de coût en termes d'actions de compteurs. En effet, ces notions n'ont pas d'analogue si on les projette dans la théorie des langages. Finalement, certains résultats peuvent être considérés comme « hybrides » : ils généralisent d'une certaine manière des résultats classiques, mais la spécificité des fonctions de coût se manifeste fortement. On peut citer parmi ceux-ci la congruence syntactique issue des $\omega\sharp$-expression, ainsi que la caractérisation de la classe des fonctions de coût quasi-faibles généralisant le théorème de Rabin.

Ces travaux peuvent être poursuivis dans de nombreuses directions, on en esquisse certaines ici. L'un des objectifs de la thèse était d'obtenir une meilleure compréhension des fonctions de coût régulières, et de définir plusieurs critères de classification, notamment en termes de nombres de compteurs nécessaires aux automates. En particulier on conjecture que le nombre de compteurs dans un automate déterministe en histoire (sur mots finis pour commencer) était identique dans le cas B et S. L'étude du déterminisme en histoire a pour but la description d'une forme normale d'automate de coût, qui serait auto-duale : le passage de B à S (et vice-versa) se ferait de manière syntaxique, de manière analogue à la complémentation d'un automate déterministe classique. Ce problème peut également être examiné dans un cadre classique, sans parler de compteurs, et pose déjà des difficultés dans ce cadre. On peut montrer que sur mots finis, un automate déterministe en histoire contient en réalité un automate déterministe, et qu'il suffit de retirer des transitions pour l'obtenir. Sur mots infinis et avec les automates de Büchi, ceci n'est plus vrai [Orna Kupferman et Udi Boker,

communication personnelle], mais en revanche, le pouvoir expressif des automates de Büchi déterministes en histoire est le même que celui des automates de Büchi déterministes. La question suivante est donc naturelle :

Problème ouvert : Soit $\mathcal{A}$ un automate de Büchi déterministe en histoire. Quelle est la complexité en nombre d'états de la déterminisation de $\mathcal{A}$ dans la classe des automates de Büchi ? Est-ce linéaire, polynomial, ou exponentiel dans le cas général ?

Le point de vue algébrique des fonctions de coût a été développé ici, mais beaucoup de choses restent à creuser. Il y a par exemple un travail à faire pour relier précisément ce travail avec le point de vue des mots profinis introduit dans [Tor11]. Du point de vue des mots infinis, on n'a pas développé ici la théorie algébrique comme on l'a fait sur les mots finis, une première complétion de ce travail consisterait donc par exemple à étendre aux mots infinis la congruence syntactique, ainsi que la caractérisation algébrique de la classe des fonctions de coût reconnues par CFO.

Le principal problème ouvert de la théorie des fonctions de coût reste la décidabilité de l'existence de borne pour les B-automates à parité sur arbres infinis. On a vu que d'après [CL08], résoudre ce problème permettrait de résoudre du même coup celui de l'indice de Mostowski non-déterministe, qui constitue un problème ouvert célèbre en théorie des langages.

On a ici complètement laissé de côté les arbres finis. Ce choix s'explique plus par des concours de circonstances au long de la thèse que par une décision réfléchie. Il est à noter que contrairement aux arbres infinis, l'équivalence entre B- et S-automates a été obtenue sur les arbres finis [CL10], et on peut donc parler de fonctions de coût régulières dans ce contexte. La généralisation des résultats de la thèse des mots finis aux arbres finis offre une piste de recherche à explorer. En particulier ceci permettrait de mesurer la dépendance des résultats aux outils spécifiques aux mots finis. Par exemple est-il toujours possible de décider si une fonction de coût régulière sur arbres finis est temporelle si l'on ne dispose pas de la notion de semigroupe de stabilisation, difficilement généralisable aux arbres ?

On peut ensuite aller plus loin, en examinant la notion de fonction de coût algébrique, possiblement définie par des formes généralisées d'automates à pile ou de grammaires. On sait par exemple qu'il est possible de décider si un S-automate à pile est borné, en réduisant ce problème à celui de l'existence d'une borne pour les automates de coût sur arbres finis, résolu dans [CL10].

Le point de vue adopté dans cette thèse sur la théorie des fonctions de coût est volontairement théorique. Une prolongation naturelle de ce travail est l'étude du versant plus pratique, à commencer par une étude de complexité des algorithmes décrits dans les preuves de décidabilité et de conversion entre les différents formalismes. On a déjà entamé ce processus, avec la description d'un algorithme polynomial en espace qui permet de décider si une formule de CLTL définit une fonction de coût bornée. On pourrait enfin implémenter ces algorithmes, et explorer les champs d'application éventuels de la théorie des fonctions de coût régulières.

Bibliographie

[AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. Assoc. Comput. Mach.*, 41(1) :181–204, 1994.

[AKY08] Parosh Aziz Abdulla, Pavel Krcal, and Wang Yi. R-automata. In *CONCUR 2008—concurrency theory*, volume 5201 of *Lecture Notes in Comput. Sci.*, pages 67–81, Berlin, 2008. Springer.

[Arn85] André Arnold. A syntactic congruence for rational ω-languages. *Theoret. Comput. Sci.*, 39(2-3) :333–335, 1985.

[Bal04] Sebastian Bala. Regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. In *STACS 2004*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 596–607. Springer, Berlin, 2004.

[BC06] Mikołaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *Proceedings of LICS 2006*, pages 285–296. IEEE Computer Society Press, 2006.

[Boj04a] Mikołaj Bojańczyk. A bounding quantifier. In *Computer science logic*, volume 3210 of *Lecture Notes in Comput. Sci.*, pages 41–55, Berlin, 2004. Springer.

[Boj04b] Mikołaj Bojańczyk. A bounding quantifier. In *Computer science logic*, volume 3210 of *Lecture Notes in Comput. Sci.*, pages 41–55. Springer, Berlin, 2004.

[BOW09] Achim Blumensath, Martin Otto, and Mark Weyer. Boundedness of monadic second-order formulae over finite words. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 67–78, Berlin, 2009. Springer.

[BT09] Mikolaj Bojanczyk and Szymon Torunczyk. Deterministic automata and extensions of weak mso. In *FSTTCS*, pages 73–84, 2009.

[BT12] Mikolaj Bojanczyk and Szymon Torunczyk. Weak mso+u over infinite trees. In *STACS*, pages 648–660, 2012.

[Büc62] 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.

[CK93] Karel Culik, II and Jarkko Kari. Image compression using weighted finite automata. In *Mathematical foundations of computer science 1993 (Gdańsk, 1993)*, volume 711 of *Lecture Notes in Comput. Sci.*, pages 392–402. Springer, Berlin, 1993.

[CKL10] Thomas Colcombet, Denis Kuperberg, and Sylvain Lombardy. Regular temporal cost functions. In *Automata, languages and programming. Part II*, volume 6199 of *Lecture Notes in Comput. Sci.*, pages 563–574. Springer, Berlin, 2010.

[CL08] Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *Automata, languages and programming. Part II*, volume 5126 of *Lecture Notes in Comput. Sci.*, pages 398–409. Springer, Berlin, 2008.

[CL10] Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *25th Annual IEEE Symposium on Logic in Computer Science LICS 2010*, pages 70–79. IEEE Computer Soc., Los Alamitos, CA, 2010.

[Col07a] Thomas Colcombet. A combinatorial theorem for trees : applications to monadic logic and infinite structures. In *Automata, languages and programming*, volume 4596 of *Lecture Notes in Comput. Sci.*, pages 901–912, Berlin, 2007. Springer.

[Col07b] Thomas Colcombet. Factorisation forests for infinite words. In *FCT*, pages 226–237, 2007.

[Col09] Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150, Berlin, 2009. Springer.

[Col11] Thomas Colcombet. Regular cost functions, part I : logic and algebra over words. Submitted, Special issue of ICALP09, 2011.

[DG07] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoret. Comput. Sci.*, 380(1-2) :69–86, 2007.

[DG08] Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and automata*, volume 2 of *Texts Log. Games*, pages 261–306. Amsterdam Univ. Press, Amsterdam, 2008.

[DG10] Stéphane Demri and Paul Gastin. Specification and verification using temporal logics. In *Modern applications of automata theory*, volume 2 of *IISc Research Monographs*. World Scientific, 2010. To appear.

[Egg63] L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10 :385–397, 1963.

[Eil74] Samuel Eilenberg. *Automata, languages, and machines. Vol. A*. Academic Press [A subsidiary of Harcourt Brace Jovanovich, Publishers], New York, 1974. Pure and Applied Mathematics, Vol. 58.

[FGO12] Nathanaël Fijalkow, Hugo Gimbert, and Youssouf Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. In *LICS*, 2012.

[GHR94] Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal logic. Vol. 1*, volume 28 of *Oxford Logic Guides*. The Clarendon Press Oxford University Press, New York, 1994. Mathematical foundations and computational aspects, Oxford Science Publications.

[Gre51] J. A. Green. On the structure of semigroups. *Ann. of Math. (2)*, 54 :163–172, 1951.

[GT07] Gösta Grahne and Alex Thomo. Boundedness of regular path queries in data integration systems. In *IDEAS*, pages 85–92, 2007.

[Has79] Kosaburo Hashiguchi. A decision procedure for the order of regular events. *Theoret. Comput. Sci.*, 8(1) :69–72, 1979.

[Has82a] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. System Sci.*, 24(2) :233–244, 1982.

[Has82b] K. Hashiguchi. Regular languages of star height one. *Inform. and Control*, 53(3) :199–210, 1982.

[Has83] K. Hashiguchi. Representation theorems on regular languages. *J. Comput. System Sci.*, 27(1) :101–115, 1983.

[Has88] Kosaburo Hashiguchi. Relative star height, star height and finite automata with distance functions. In *Formal Properties of Finite Automata and Applications*, pages 74–88, 1988.

[Has90] Kosaburo Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theoret. Comput. Sci.*, 72(1) :27–38, 1990.

[HR05] Ian M. Hodkinson and Mark Reynolds. Separation - past, present, and future. In *We Will Show Them ! (2)*, pages 117–142, 2005.

[Kam68] Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. Phd thesis, University of Warsaw, 1968.

[Kir04] Daniel Kirsten. Desert automata and the finite substitution problem (extended abstract). In *STACS 2004*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 305–316. Springer, Berlin, 2004.

[Kir05] Daniel Kirsten. Distance desert automata and the star height problem. *Theor. Inform. Appl.*, 39(3) :455–509, 2005.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata studies*, Annals of mathematics studies, no. 34, pages 3–41. Princeton University Press, Princeton, N. J., 1956.

[KPV09] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2) :83–103, 2009.

[Kup11] Denis Kuperberg. Linear temporal logic for regular cost functions. In *28th International Symposium on Theoretical Aspects of Computer Science*, volume 9 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 627–636. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2011.

[KV99] Orna Kupferman and Moshe Y. Vardi. The weakness of self-complementation. In *STACS 99 (Trier)*, volume 1563 of *Lecture Notes in Comput. Sci.*, pages 455–466. Springer, Berlin, 1999.

[KV01] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3) :408–429, 2001.

[KV11] Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata : a new variant of weakness. In *31st International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 13 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 66–77. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2011.

[KV12] Denis Kuperberg and Michael Vanden Boom. On the expressive power of cost logics over infinite words. In *ICALP (2)*, pages 287–298, 2012.

[LMP10] François Laroussinie, Antoine Meyer, and Eudes Petonnet. Counting ctl. In *Foundations of software science and computational structures*, volume 6014 of *Lecture Notes in Comput. Sci.*, pages 206–220. Springer, Berlin, 2010.

[LP04] Hing Leung and Viktor Podolskiy. The limitedness problem on distance automata : Hashiguchi's method revisited. *Theoret. Comput. Sci.*, 310(1-3) :147–158, 2004.

[LT00] Christof Löding and Wolfgang Thomas. Alternating automata and logics over infinite words. In *IFIP TCS*, pages 521–535, 2000.

[LV92] Nancy Lynch and Frits Vaandrager. Forward and backward simulations for timing-based systems. In *Real-time : theory in practice (Mook, 1991)*, volume 600 of *Lecture Notes in Comput. Sci.*, pages 397–446, Berlin, 1992. Springer.

[Moh05] Mehryar Mohri. Statistical natural language processing. In M. Lothaire, editor, *Applied Combinatorics on Words*. Cambridge University Press, 2005.

[MS95] 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.

[MSS92] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theoret. Comput. Sci.*, 97(2) :233–244, 1992.

[Niw86] Damian Niwiński. On fixed-point clones (extended abstract). In *Automata, languages and programming (Rennes, 1986)*, volume 226 of *Lecture Notes in Comput. Sci.*, pages 464–473. Springer, Berlin, 1986.

[NW05] Damian Niwiński and Igor Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. In *Proceedings of the 11th Workshop on Logic, Language, Information and Computation (WoLLIC 2004)*, volume 123 of *Electron. Notes Theor. Comput. Sci.*, pages 195–208 (electronic), Amsterdam, 2005. Elsevier.

[PP95] 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.

[PW95] Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. In *22th ICALP*, number 944 in Lecture Notes in Computer Science, pages 348–359, Berlin, 1995. Springer.

[Rab63] Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3) :230–245, 1963.

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141 :1–35, 1969.

[Rab70] Michael O. Rabin. Weakly definable relations and special automata. In *Mathematical Logic and Foundations of Set Theory (Proc. Internat. Colloq., Jerusalem, 1968)*, pages 1–23. North-Holland, Amsterdam, 1970.

[RS59] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Develop.*, 3 :114–125, 1959.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. Assoc. Comput. Mach.*, 32(3) :733–749, 1985.

[Sch61] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4 :245–270, 1961.

[Sch65] M.-P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control 8*, pages 190–194, 1965.

[Sim78] Imre Simon. Limited subsets of a free monoid. In *19th Annual Symposium on Foundations of Computer Science (Ann Arbor, Mich., 1978)*, pages 143–150. IEEE, Long Beach, Calif., 1978.

[Sim90] Imre Simon. Factorization forests of finite height. *Theoret. Comput. Sci.*, 72(1) :65–94, 1990.

[Sim94] Imre Simon. On semigroups of matrices over the tropical semiring. *RAIRO Inform. Théor. Appl.*, 28(3-4) :277–294, 1994.

[Tor11] Szymon Toruńczyk. *Languages of profinite words and the limitedness problem.* Phd thesis, Computer Science Department, University of California at Los Angeles, USA, 2011.

[VB11] Michael Vanden Boom. Weak cost monadic logic over infinite trees. In *Mathematical foundations of computer science 2011*, volume 6907 of *Lecture Notes in Comput. Sci.*, pages 580–591. Springer, Heidelberg, 2011.

[Wil99] Thomas Wilke. Classifying discrete temporal properties. In Christoph Meinel and Sophie Tison, editors, *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1999.