N° d'ordre: 3270

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Antoine Meyer

Équipes d'accueil :	Groupe Galion (Irisa)
	Équipe Modélisation et Vérification (Liafa – Université de Paris 7)
École Doctorale :	Matisse
Composante universitaire :	IFSIC

Titre de la thèse :

Graphes infinis de présentation finie

Soutenue le 14 octobre 2005 en salle des thèses devant la commission d'examen

M. :	Jean-Claude	Raoult	Président
MM. :	Kamal	Lodaya	Rapporteurs
	Denis	Lugiez	
MM. :	Colin	Stirling	Examinateurs
	Wolfgang	Thomas	
	Ahmed	Bouajjani	
	Didier	Caucal	

Remerciements

Je tiens tout d'abord à remercier les membres de mon jury, qui m'ont fait le grand honneur de bien vouloir consacrer une partie de leur temps à l'évaluation de ce travail.

Merci à Denis Lugiez d'avoir bien voulu évaluer ce document en pleine période de vacances. J'espère que la lecture que je lui ai imposée ne lui laissera pas un trop mauvais souvenir.

My sincerest thanks to Kamal Lodaya for agreeing to review this work. Thank you also, Kamal, for rich scientific contacts throughout these three years, and for giving me the occasion to investigate the field of Indian theoretical computer science from a closer perspective.

My thanks to Colin Stirling, whose name (although we met only twice) is very familiar to me, doubtless because of his work on rational graphs which is one of my earliest references in the domain.

Ich bedanke mich sehr herzlich bei Wolfgang Thomas für seine Teilnahme an meiner Prüfungskommission und für seine hilfreiche Unterstützung meiner Arbeit.

Un grand merci enfin à Jean-Claude Raoult d'avoir accepté de présider ce jury, et pour son accueil au sein de l'équipe Galion dès mes premiers jours de recherche acharnée.

J'ai eu au cours de cette thèse le plaisir rare de côtoyer non pas un, mais deux excellents chercheurs qui m'ont encadré et conseillé.

Merci à Ahmed Bouajjani de m'avoir accueilli au Liafa, d'être un si bon directeur de thèse, et merci pour sa grande humanité. Merci aussi pour les bonnes adresses de restaurants parisiens.

Avoir rencontré Didier Caucal l'année de mon DEA représente à mes yeux un grand privilège, et aussi certainement la cause de *bien des* bouleversements dans l'orientation de ma vie professionnelle et, par la force des choses, personnelle (rassure toi Didier, je n'en regrette pas un seul). Je lui suis extrêmement reconnaissant de sa façon de faire de la recherche, de sa conception de l'enseignement et de l'encadrement, et de sa sympathie.

Je souhaite faire part de toute ma gratitude envers les personnes avec qui j'ai eu l'occasion de travailler ou d'échanger des idées pendant ces 36 mois.

Tout d'abord, je tiens à exprimer à Arnaud Carayol mes plus sincères remerciements pour son amitié, pour son soutien et pour son excellent travail. Cette thèse, dont une partie des résultats est issue d'une collaboration avec Arnaud, doit beaucoup à sa clairvoyance et a ses nombreuses et excellentes idées, mais aussi tout simplement à son soutien, à son accueil chaleureux et à sa bonne humeur. Je souhaite de tout cœur que nos échanges durent encore longtemps.

Many thanks to Mooly Sagiv for inviting and welcoming me at the university of Tel-Aviv. I am very grateful to him and to Greta Yorsh and Alex Rabinovitch for a very interesting collaboration, and a unique opportunity to walk the ground of one of the most intriguing countries there is.

Sincères remerciements aux ressortissants de l'Insa, Julien, Elisa, Olivier, David, Gurvan, Alban et les autres, qui m'ont conservé leur précieuse amitié et ont accepté mes passages un peu imprévisibles dans les couloirs de l'Irisa avec une patience et un flegme louables, ainsi qu'à mes amis et collègues du groupe Galion, de l'Irisa et du Liafa, et en particulier à Tanguy, Thomas et Thomas, Julien, Thierry, Emmanuelle, Christophe et Chloé, à Guillaume, Sabine et Corentin, et à Hugo, Emmanuel, Laura, Claire et Florian, à Peter, Pierre et Petr, à Tomás, à Anca, à Michel et Sam et les autres voisins de PPS, et à tous ceux que j'oublie.

Merci à Maxence d'avoir relu une partie de ma thèse. Merci aussi à Mélanie, Martine et Georges. Merci à Agnès et Patrice pour le cyber-pommier.

Merci à mes parents et à mon frère qui ne sont jamais bien loin, même quand je ne sais plus trop où j'habite.

Merci enfin à Julie, sans qui tout cela serait infiniment moins intéressant.
Merci merci merci.

Avertissement orthographique.

Pour une raison qui échappe un peu à l'auteur de ces lignes, il a finalement été décidé au dernier moment de céder à l'orthographe la plus courante, et de réécrire toutes les instances de toutes les formes du verbe « récrire » en « réécrire ».

Les lecteurs éventuellement rebutés par ce choix me feront, j'en suis sûr, la faveur d'appliquer mentalement à l'ensemble du texte le système de réréécriture fini de thèse

$$\{(\text{réécr}, \text{récr})\}.$$

Table des matières

Introduction

Cette thèse s'inscrit dans une étude systématique de familles d'objets mathématiques infinis *de présentation finie*, en l'occurrence des *graphes infinis*. Une attention particulière est vouée aux propriétés *structurelles* de chaque famille, ainsi qu'à leur comparaison. Outre son intérêt purement théorique, l'un des objectifs de ce travail est, à long terme, de proposer de nouvelles familles de modèles infinis pour des objets informatiques comme les programmes, vers un usage possible en vérification par exemple.

Le *Model Checking* vérifie des modèles

Une formulation très générale du problème de la vérification de programmes informatiques ou de systèmes similaires est : « Le comportement du programme X respecte-t-il la propriété Y ? » où Y fait référence à la correction des calculs effectués par X, au fait que X ne se bloque pas ou n'entre pas dans une boucle d'exécution infinie, ou à tout autre comportement attendu ou indésirable de X. Nous nous intéressons principalement à des propriétés qui ne dépendent pas du langage de programmation ou des détails d'implémentation de X, mais font uniquement état de la structure de ses exécutions. Dans la plupart des cas, un informaticien se repose sur ses talents de programmation, sur quelques tests dans des cas typiques, ou lorsque c'est possible sur une preuve formelle à la main, pour s'assurer que son programme est correct.

Dans certains cas, il est souhaitable (ou indispensable) de pouvoir effectuer une vérification plus systématique de la correction d'un programme par rapport à se spécification. Pour répondre à ce besoin, plusieurs thèmes de recherches voués à l'amélioration de la correction des programmes ont vu le jour. On peut mentionner la création automatisée de bancs d'essais (techniques de test, ou *testing*) [Mye79], la preuve assistée (*theorem-proving*) [NPW02, BC04, ORS92], la programmation avec preuve de correction intégrée (*proof-carrying code*) [Nec97], l'interprétation abstraite [CC77] ou divers autres types d'analyses de programmes statiques ou dynamiques.

Le présent travail a des liens particuliers avec une autre approche bien connue, la *vérification automatique*, qui tente de démontrer des propriétés de manière

algorithmique, c'est à dire sans intervention de l'utilisateur. Dans ce cadre, un programme X est associé à un modèle[1] M, qui est une structure logique donnant une description formelle de son comportement (ou de sa sémantique). D'autre part, une propriété Y à vérifier sur X est exprimée formellement par une formule ϕ exprimée dans un langage logique interprétable sur la signature de la structure M. Par exemple, le modèle associé à un programme pourrait être un automate fini avec des sommets étiquetés (ou plus exactement la structure logique sous-jacente), et le langage utilisé pour écrire ϕ pourrait être la logique temporelle LTL. Le problème de la vérification de X par rapport à Y est ici vu comme un problème de *vérification de modèle* (en Anglais *model checking*)[2] :

$$\text{Le programme } X \text{ a la propriété } Y \iff M \models \phi.$$

On parle aussi du problème de satisfaction de ϕ par M. Bien sûr, cette équivalence doit être prise avec le recul nécessaire. Pour qu'elle soit valide, il faut s'assurer que M représente fidèlement le comportement de X et que ϕ exprime bien la propriété Y, ce qui est un problème de modélisation. De plus, en admettant que ce soit effectivement le cas, une conséquence bien connue des travaux de Turing et Church [Tur36, Chu36] est que pour toutes classes suffisamment générales de modèles et de propriétés, ce problème est indécidable.

Partant de cette observation, il existe deux principales alternatives. La première est de ne considérer que des classes restreintes de programmes et de propriétés dont des modèles ayant un problème de satisfaction décidable sont connus. L'autre possibilité est d'*abstraire* les programmes trop généraux afin de pouvoir les représenter par des modèles que l'on sait traiter. Une abstraction courante est de ne pas prendre en compte toutes les valeurs possibles des variables d'un programme, mais de se restreindre à un ensemble fini de valeurs. Dans ce dernier cas, cependant, en raison de l'abstraction, la satisfaction d'une formule sur un modèle n'implique pas nécessairement la correction du programme initial. Plusieurs techniques permettant malgré tout de déduire des propriétés des programmes de l'étude de leurs modèles abstraits ont été développées, par exemple en démontrant que toute propriété du modèle abstrait est nécessairement aussi vérifiée par le programme concret, ou réciproquement (on parle d'abstraction *sûre*).

Les automates comme modèles de programmes

Des techniques de vérification automatique bien établies et d'efficacité reconnue utilisant la théorie des automates finis comme cadre général pour la modélisa-

1. Dans cette phrase, le terme *modèle* fait référence à la formalisation mathématique d'un objet concret, ici un programme.

2. En logique, un *modèle* d'une formule donnée est une structure sur laquelle la formule est vraie. Ce sens diffère légèrement du sens rencontré précédemment.

tion et la vérification de programmes ont été proposées par plusieurs auteurs (dont par exemple [CES86, VW86]). Les automates finis de mots ou d'arbres jouissent d'intéressantes propriétés de décidabilité et d'une étude approfondie comme l'un des outils principaux de la théorie des langages et d'autres domaines proches (voir par exemple [Ber79, Sak03]). Plus important, de nombreux algorithmes de complexité raisonnable ont été adaptés ou développés pour résoudre leur problème de model checking dans divers cas. Enfin, plusieurs techniques standards pour l'abstraction sûre de programmes sous forme d'automates finis ont été proposées.

Cependant, un inconvénient de telles techniques de vérification « à ensemble d'états fini » est que la classe de programmes qu'elles permettent de traiter est relativement restreinte. Même utilisés comme abstractions de programmes, les automates finis présentent une sérieuse limite d'expressivité étant donné que toute source de comportement infini des programmes doit être abstraite : la présence de variables sur des domaines arbitrairement grands, les aspects temporels (chronométrage de certains événements), le parallélisme, la création dynamique de processus ou d'autres aspects forçant la considération d'un ensemble infini de situations possibles sont autant de paramètres que les automates finis sont incapables de modéliser. Par conséquent, on ne peut espérer vérifier uniquement à l'aide d'automates finis les propriétés qui reposent spécifiquement sur ces aspects infinis.

Pour répondre à cette limitation, des avancées récentes dans le domaine de la vérification s'intéressent de plus en plus à l'élaboration de nouvelles techniques sur des objets infinis. Une des principales idées permettant d'étayer ces techniques est d'utiliser des formalismes plus généraux, certains empruntés à des domaines connexes, comme les automates à pile, les automates possédant des compteurs ou des horloges, les automates communiquants, les systèmes de réécriture, etc. Cependant, une difficulté inhérente à ce type de formalismes à ensembles de configurations infinis est qu'il est moins aisé de visualiser la structure de leurs exécutions que dans le cas fini. Un moyen convaincant de faire face à cet inconvénient est d'étudier les familles de *graphes infinis* engendrés par ces formalismes à espace de configurations infini.

Graphes infinis

Il existe plusieurs façons de définir un graphe infini associé à la description finie d'une machine quelconque. Dans [MS85], Muller et Schupp considèrent la famille des *graphes d'automates à pile*, dont les sommets représentent les configurations accessibles depuis l'état initial dans un automate à pile, et dont les arcs illustrent l'effet des transitions de l'automate. Une caractéristique fondamentale de leur approche fut de proposer une caractérisation différente de cette famille de graphes, indépendante des valeurs proprement dites des configurations : ils ont

démontré que l'ensemble de tous les graphes connexes que l'on peut obtenir en retirant des « tranches » successives à un graphe d'automate à pile à partir d'un sommet quelconque est fini. Cette propriété de décomposition finie par distance leur permit, en utilisant le résultat de Rabin [Rab69] de décidabilité de la théorie du second ordre monadique (MSO) de l'arbre binaire complet (infini), d'étendre ce résultat de décidabilité aux graphes d'automates à pile, ou en d'autre termes de démontrer que les graphes d'automates à pile ont un problème de model-checking décidable pour la logique MSO.

Dans [Cou90], Courcelle a démontré le même résultat pour la famille strictement plus générale des graphes *HR-équationnels*, qui sont les graphes engendrés par des grammaires déterministes de graphes, une fois encore en se ramenant au théorème de Rabin. Dans certains de ses travaux [Cau92, Cau96], Caucal a décrit deux caractérisations possibles des graphes d'automates à pile et des graphes HR-équationnels, l'une utilisant des systèmes de réécriture et l'autre utilisant des transformations de l'arbre binaire complet préservant la décidabilité de la théorie du second ordre monadique. Cette dernière présentation lui permit en particulier de proposer une nouvelle preuve de la décidabilité de MSO sur ces graphes, ainsi qu'une extension de ce résultat à la famille strictement plus générale des graphes *préfixe-reconnaissables*. Dans les années qui suivirent, plusieurs autres familles de graphes furent définies et étudiées à l'aide de caractérisations diverses, par exemple les graphes *automatiques* [BG00], les graphes rationnels [Mor00], les graphes préfixe-reconnaissables d'ordre supérieur [CK02a] et diverses familles de graphes de réécriture de termes.

Ces résultats mettent en lumière l'intérêt de considérer les graphes infinis *à isomorphisme près*, c'est à dire sans imposer de nommage particulier de leurs sommets, mais en ne conservant et en n'étudiant que leur *structure*. Ceci peut être expliqué par plusieurs arguments simples. Le plus intuitif est que l'on ne souhaiterait pas considérer comme différents deux graphes dont la seule différence réside dans le choix d'un alphabet pour leurs sommets si ceux-ci sont représentés par des mots ou des termes, ou par le choix d'une base numérique s'ils sont représentés par des nombres.

D'un point de vue plus concret, considérons deux graphes modélisant le comportement de deux programmes réactifs recevant des commandes d'un utilisateur ou d'un environnement. Les sommets de tels graphes représentent l'état interne du programme correspondant, incluant les variables, la pile d'appels ou le pointeur de programme, alors qu'il attend une entrée. Les arcs représentent l'effet de chaque entrée possible sur cet état. Si l'on s'intéresse par exemple à déterminer si ces deux programmes ont le même comportement (ou simplement à dessiner les exécutions possibles de chacun d'entre eux), il est inutile de comparer leurs détails concrets d'implémentation, dont certaines différences peuvent ne pas être significatives, par exemple s'il s'agit de deux programmes identiques écrits dans

deux langages de programmation différents.

Dans un cadre théorique, une conséquence directe de la vision à isomorphisme près est que la même famille de graphes peut avoir plusieurs ensembles de représentants (ou nommages possibles). Plusieurs questions naturelles en découlent. L'une est de déterminer si deux formalismes donnés décrivent en fait la même famille de graphes, ou de comparer des familles de représentants entre elles. Une autre question importante est de déduire des propriétés d'une famille de graphes de celles de ses familles de représentant (comme par exemple dans le cas des graphes d'automates à pile). Enfin, puisque les graphes infinis sont si aisément mis en rapport avec d'autres domaines de l'informatique, on peut espérer que leur étude apportera de nouveaux résultats et applications dans des domaines connexes, comme la théorie des langages, la complexité, ou la vérification de programmes.

Plan de la thèse

Introduction au domaine

Après avoir présenté quelques notions indispensables sur les langages, les automates, les relations et la logique dans le chapitre 1, nous donnons dans le chapitre 2 un bref survol de l'état de l'art concernant les graphes infinis. Ce chapitre commence par présenter plusieurs techniques permettant de décrire la structure de graphes infinis à l'aide de formalismes finis, soit de façon interne en spécifiant leurs arcs explicitement, soit de façon externe en décrivant directement leur structure par des séries de transformations à partir de graphes plus simples, soit à l'aide d'autre types de mécanismes finis. Nous expliquons aussi certaines des spécificités de chacune de ces caractérisations.

Dans un second temps, nous présentons certaines de familles de graphes les plus connues et le mieux étudiées. Nous rappelons tout d'abord les définitions de familles de graphes ayant une théorie monadique décidable. Ceci inclut bien sûr les graphes d'automates à pile de Muller et Schupp [MS85], les graphes HR et VR-équationnels de Courcelle [Cou90] et la hiérarchie préfixe-reconnaissable de Caucal [Cau96, CK02a]. La plupart des résultats de décidabilité de la théorie monadique du second ordre sur ces graphes font usage des théorèmes similaires de Büchi sur la demie-droite [Büc62] et de Rabin sur l'arbre binaire complet [Rab69]. Ensuite, nous introduisons les graphes rationnels étudiés en premier par Morvan [Mor00], et certaines de leurs sous-familles comme les graphes automatiques [BG00]. Enfin, nous mentionnons les graphes associés aux machines de Turing [Cau03], et une extension possible de certaines de ces familles aux graphes dont les sommets sont des termes (comme par exemple les graphes de réécriture des systèmes clos de termes [Löd02] ou les graphes terme-automatiques [BG00]).

Nous concluons par quelques mots à propos des directions de recherche actuelles en matière de vérification de systèmes à espace d'états infinis, et de leurs liens avec la théorie des graphes infinis.

Systèmes de réécriture de termes à dérivation rationnelle

Le chapitre 3 est consacré à la caractérisation de familles de systèmes de réécriture de termes dont la relation de dérivation (la relation formée de toutes les paires de termes dont le premier composant peut être réécrit en le second en un nombre quelconque d'étapes) est récursive et peut être représentée de façon finie. Nous étendons au cas des termes une classification des systèmes de réécriture de mots proposée dans [Cau00], qui s'appuie sur les différentes manières dont parties gauches et parties droites de règles peuvent se chevaucher. Nous montrons que, vis-à-vis d'une notion de rationalité sur les n-uplets de termes, cette classification permet de caractériser trois familles de systèmes de réécriture dont les relations de dérivation sont rationnelles. Ces résultats ont également pour conséquence des propriétés de préservation effective des ensembles réguliers de termes.

De nombreux travaux s'intéressent aux systèmes de réécriture de termes. Parmi les plus proches de notre point de vue, nous pouvons citer par exemple [GV98] et [TKS00], qui s'intéressent spécifiquement à la préservation des ensembles réguliers de termes par différents systèmes de réécriture. En ce qui concerne les relations de dérivation en tant que telles, Dauchet et Tison se sont intéressés aux systèmes de réécriture clos, qui sont les systèmes dont les règles ne contiennent aucune variable [DT85]. Ils ont en particulier caractérisé les relations de dérivation de ces systèmes à l'aide d'automates d'arbres, ce qui a pour conséquence que les systèmes clos ont une théorie du premier ordre (avec accessibilité) décidable [DHLT90]. Dans la lignée de ces travaux, [Löd02] et [Col02] ont proposé une étude structurelle plus approfondie des graphes de réécriture des systèmes clos, et l'ont située par rapport à d'autres familles de graphes infinis.

Une application possible de nos résultats existe dans le domaine du model checking symbolique, dont l'idée principale est de représenter des ensembles potentiellement infinis de configurations d'un système par des ensembles réguliers de mots, et les transitions par des règles de réécriture ou par des transducteurs (voir par exemple [BJNT00]). Cet axe de recherche est actuellement étendu à des systèmes dont les configurations ont des configurations plus riches, par exemple arborescentes [AJMD02, BT02]. Cette approche repose directement sur la capacité à calculer l'ensemble de successeurs ou des prédécesseurs d'un ensemble de configurations donné, par exemple un langage régulier de termes.

Le travail contenu dans ce chapitre a été présenté à la conférence FoSSaCS 2004 à Barcelone et publié dans ses actes [Mey04]. Une version longue de ce article a récemment été soumis pour publication en revue, et est en cours de relecture.

Automates infinis pour les langages contextuels

Le chapitre 4 présente une étude approfondie de deux familles de graphes vus comme des automates « temps réel » pour les langages contextuels, au sens où ils ne contiennent pas d'arcs étiquetés par ε.

Dans un premier temps, nous étudions la famille des graphes rationnels et certaines de ses sous-familles. Nous proposons une nouvelle démonstration du fait que les traces de ces graphes sont les langages contextuels, résultats originellement démontrés par [MS01, Ris02][3]. Le principal avantage de notre démonstration est qu'elle n'utilise pas de résultats antérieurs et se lit de façon indépendante, ce qui n'était pas le cas de la démonstration initiale. Nous considérons aussi quelques restrictions structurelles pertinentes sur le degré ou le nombre de sommets initiaux de tels graphes, et examinons les conséquences quant à leurs langages.

Dans un second temps, nous introduisons la famille des graphes linéairement bornés comme les graphes de transition des machines linéairement bornées, qui sont les accepteurs les plus connus des langages contextuels. Par conséquent, les traces de ces graphes sont également les langages contextuels. Nous fournissons deux autres caractérisations de cette famille et la comparons avec des travaux proches existants [KNU02, Cau03].

Dans les deux cas, nous nous intéressons tout particulièrement à la famille des langages contextuels déterministes, et caractérisons des sous-familles de chacune de ces familles de graphes dont les traces sont précisément ces langages. Il est intéressant de noter que le problème de l'équivalence des langages contextuels déterministes et non-déterministes [Kur64] n'est à ce jour pas résolu. Nous concluons ce chapitre par une comparaison des deux familles de graphes, dont il ressort qu'elles sont incomparables en général, mais que malgré leur nature fort différente, dans le cas borné les graphes rationnels forment une sous-famille stricte des graphes linéairement bornés.

Les résultats présentés dans ce chapitre sont issus d'un travail commun avec Arnaud Carayol. Ils ont été soumis à une publication en revue [CM05a, CM05c], et sont actuellement en attente d'acceptation. Une version plus courte de [CM05c] a été présentée à la conférence MFCS 2005 (Gdansk) et publié dans les actes de cette conférence [CM05b].

Analyse d'accessibilité de processus hors-contexte d'ordre supérieur

Enfin, nous présentons au chapitre 5 une méthode directe d'analyse symbolique d'accessibilité pour une famille d'automates à pile d'ordre supérieur. La motivation de ce travail a trait au calcul de l'ensemble des prédécesseurs d'un

3. Une présentation révisée de ces travaux a récemment été publiée dans [MR05]

ensemble de configurations donné, représenté par un langage régulier de mots. Nous montrons que certaines des techniques utilisées dans [BEM97] pour l'analyse d'accessibilité des automates à pile peuvent être étendues à une classe que nous appelons processus hors-contexte d'ordre supérieur, qui sont l'extension naturelle des processus hors-contexte (aussi appelés algèbres de processus simples) à l'ordre supérieur. Ces classes sont obtenues en considérant des automates à pile (d'ordre supérieur) avec un unique état de contrôle.

Nous proposons comme application de ces résultats une technique de model checking symbolique sur les processus hors-contexte d'ordre supérieur pour le fragment existentiel de la logique temporelle CTL restreinte aux modalités EU et EX.

Ce travail a été réalisé en commun avec Ahmed Bouajjani. Il a été présenté en 2004 à la conférence FSTTCS à Chennai, et publié dans ses actes [BM04].

Chapitre 1

Notions préliminaires

Dans ce chapitre, nous rappelons quelques uns des notations et concepts fondamentaux auxquels il est fait appel dans ce document. Après avoir donné la définition des mots, des termes et des graphes, nous présentons quelques notions concernant les systèmes de réécriture de mots et de termes, ainsi qu'un bref survol de la hiérarchie de langages bien connue attribuée à Chomsky. En particulier, nous rappelons la définition d'une hiérarchie stricte d'accepteurs finis pour chacune de ces familles de langages. Nous présentons ensuite les définitions de plusieurs classes de relations binaires sur les mots et les termes caractérisées soit de façon « algébrique », soit par l'intermédiaire d'automates. Nous concluons ce chapitre par un bref exposé des logiques les plus courantes sur les graphes, ainsi qu'un rappel de certaines des plus courantes idées de vérification par automates.

1.1 Bestiaire

1.1.1 Mots

Nous considérons des ensembles finis de symboles, ou *lettres*, appelés *alphabets*. Les séquences finies de lettres sont appelées *mots*, et les ensembles de mots *langages*. Dans la suite, autant que faire se peut, nous utiliserons les lettres majuscules grecques Σ et Γ pour faire référence à des alphabets, les lettres latines majuscules ou minuscules a, b, $c\ldots$ pour des lettres de ces alphabets, et les minuscules u, v et w pour des mots.

Un mot u sur un alphabet Σ peut être vu comme un n-uplet $(a_1,\ldots,a_n)$ d'éléments de Σ, ou de façon équivalente comme une application de l'intervalle d'entiers $[1,n]$ dans Σ. Il est habituellement écrit $a_1 \ldots a_n$. Sa i-ème lettre est $u(i) = a_i$. L'ensemble de tous les mots sur Σ se note Σ^*. Le nombre d'occurrences de lettres dans u est sa *longueur* $|u|$ (ici $|u| = n$). L'unique mot de longueur 0, appelé *mot vide*, se note ε. La concaténation de deux mots $u = a_1 \ldots a_n$ et

$v = b_1 \dots b_m$ est le mot $uv = a_1 \dots a_n b_1 \dots b_m$. Un mot u est un *préfixe* d'un autre mot v s'il existe w tel que $uw = v$. Réciproquement, on appelle w *suffixe* de v. Tout mot préfixe de w ou suffixe de u est un *facteur* de v.

L'opération de concaténation s'étend aux ensembles de mots : pour tous $A,B \subseteq \Sigma^*$, AB fait référence à l'ensemble $\{uv \mid u \in A \text{ and } v \in B\}$. Par un léger abus de notation, on écrira souvent u le singleton $\{u\}$. De la même façon, on pourra écrire directement Σ l'ensemble des mots de longueur 1, et plus généralement Σ^n l'ensemble des mots de longueur n.

Monoïdes. La structure algébrique sous-jacente aux mots est le *monoïde*. Un monoïde M est simplement un ensemble, potentiellement infini[1], muni d'une opération interne associative, appelée produit, dont l'élément neutre 1_M appartient à l'ensemble. Une partie S d'un monoïde M est un ensemble de *générateurs libres* de M si chaque élément de M admet une décomposition unique comme produit d'éléments de S. On dit alors que M est *libre*. Si de plus S est fini, M est dit *de type fini*.

Exemple 1.1. L'ensemble $\mathbb{N}$ des entiers positifs est un monoïde libre pour l'addition. Son élément neutre est 0. Il est librement engendré par le singleton $\{1\}$ (et est donc de type fini).

Exemple 1.2. L'ensemble Σ^* de tous les mots sur un alphabet Σ est un monoïde pour la concaténation. Il est de type fini et librement engendré par Σ, son élément neutre est le mot vide ε. En théorie des langages, la caractérisation des mots comme éléments du monoïde libre est utile pour définir des familles de langages et de relations (cf. §1.4).

1.1.2 Termes

Les termes sont des objets mathématiques plus généraux que les mots, qui représentent intuitivement des séquences d'applications de fonctions de plusieurs arguments. Les termes sont définis à l'aide d'alphabets *gradués*, c'est à dire d'alphabets dont chaque symbole est associé à un unique entier positif appelé son *arité*. L'arité d'un symbole f est notée $a(f)$. Si $a(f) = n$, on précisera parfois l'arité de f par la notation $f^{(n)}$. Les symboles d'arité 0 sont appelés *constantes*.

Soit $F = \bigcup_{n \geq 0} F_n$ un alphabet gradué fini, chaque F_n étant l'ensemble des symboles de F d'arité n, et X un ensemble fini de *variables* disjoint de F (tous les ensembles F_n sont évidemment disjoints entre eux). Les variables sont considérées comme des symboles d'arité 0. L'ensemble des termes de premier ordre sur F à variables dans X, noté $T(F,X)$, est le plus petit ensemble contenant X tel que $f \in F_n \ \wedge \ t_1, \dots, t_n \in T(F,X)$ implique $f t_1 \dots t_n \in T(F,X)$. Les mots peuvent

1. Nous ne considérerons ici que des monoïdes dénombrables.

être vus comme des termes à 1 variable sur un alphabet gradué dont tous les symboles sont d'arité 1. La *hauteur* d'un terme $ft_1 \dots t_n \in T(F,X)$ est définie récursivement comme $1 + max(h_1, \dots, h_n)$, où chaque h_i est la hauteur du sous-terme t_i. Pour faciliter la lecture, un terme $ft_1 \dots t_n$ avec $a(f) = n$ sera parfois écrit $f(t_1, \dots t_n)$.

Un terme est dit *propre* ou *non trivial* s'il contient au moins un symbole de F (autrement dit, s'il n'est pas réduit à une variable). Les termes ne contenant aucune variable sont appelés *clos*. L'ensemble des termes clos se note $T(F,\emptyset)$ ou plus simplement $T(F)$. L'ensemble des variables ayant une occurrence dans un terme t est $V(t)$. Un terme t est dit *linéaire* si chacune de ses variables n'a qu'une seule occurrence dans t. Un terme linéaire t à n variables est appelé n-contexte, et ses variables sont parfois notées $\Box_1, \dots, \Box_n$, lues de la gauche vers la droite dans leur ordre d'occurrence. Notez que chaque $\Box_i$ n'est pas forcément le nom réel de la i-ème variable de t, mais seulement un raccourci d'écriture. L'ensemble des n-contextes sur F est noté $C_n(F)$, l'ensemble de tous les contextes $C(F)$.

Une opération courante sur les termes est la *substitution*. Une substitution est entièrement définie par une application σ de X dans $T(F,X)$, et étendue par morphisme à $T(F,X)$: on note $t\sigma$ l'application de σ à un terme t, dont le résultat est obtenu en remplaçant chaque occurrence de chaque variable x dans t par le terme $\sigma(x)$. L'ensemble des substitutions sur F et X est noté $S(F,X)$. On utilise la notation spéciale $c[t]$ pour la substitution par t de l'unique variable d'un 1-contexte c. On dit alors que c est un *préfixe* et t un *suffixe* de $c[t]$, par analogie avec les mots. S'il existe une substitution σ telle que $t = t'\sigma$, on dit de plus que t' est un *facteur* de $c[t] = c[t'\sigma]$.

Positions. Soit $\mathbb{N}$ l'ensemble des entiers strictement positifs, on appelle *position* tout mot de $\mathbb{N}^*$. De même que les mots peuvent être vus comme des applications des intervalles d'entiers vers les alphabets, chaque terme t dans $T(F,X)$ peut être représenté par une application d'un ensemble de positions fermé par préfixe $Pos(t)$, appelé le *domaine* du terme, vers l'ensemble $F \cup X$. Soit $t = ft_1 \dots t_n$ un terme, ε est la position du premier symbole fonctionnel f dans t, et pour tous $k \in [1,n]$, $p \in \mathbb{N}^*$, la position kp fait référence au symbole à la position p dans t_k. On écrit $t(p) = f$ lorsque le symbole à la position p dans t est f. On note $pos(x,t)$ l'ensemble des positions auxquelles la variable $x \in X$ apparaît dans le terme $t \in T(F,X)$.

Dans la suite, nous utiliserons l'ordre partiel préfixe sur les positions, noté $\geqslant$. Soient p et q deux positions, on a $p \geqslant q$ s'il existe $q' \in \mathbb{N}^*$ tel que $p = qq'$. Si de plus $q' \neq \varepsilon$, on écrit $p > q$.

Mots de termes. Par analogie avec les mots, on appelle *mots de termes* les séquences de termes. L'ensemble des mots de termes sur F et X est noté $T^*(F,X)$.

Toutes les définitions précédentes (termes clos, linéarité, contextes, substitutions, préfixes, suffixes etc.) s'étendent naturellement aux mots de termes. De plus, pour tout mot de termes $s = s_1 \dots s_n$ et tout n-contexte t, on utilise la notation courte $t[s]$ pour désigner le mot de termes $t\sigma$, où σ est l'application $\{\Box_i \mapsto s_i \mid i \in [1,n]\}$. Rappelons que $\Box_i$ fait référence à la i-ème variable la plus à gauche dans t. Les notations s'étendent aux ensembles de mots de termes de la façon habituelle.

Exemple 1.3. Soit $F = \{f^{(2)},g^{(1)},a^{(0)},b^{(0)}\}$. On a $F_2 = \{f\}$,$F_1 = \{g\}$ et $F_0 = \{a,b\}$. Soit $t = ffab\ ga$ un terme clos dans $T(F)$, aussi écrit $f(f(a,b),g(a))$. Soient x,y deux variables, le terme $c_1 = fxy$ est un 2-contexte et $c_2 = fxga$ un 1-contexte ; tous deux appartiennent à $T(F,\{x,y\})$ et sont des préfixes de t. On peut écrire $t = c_1\sigma$ et $t = c_2\sigma$ avec $\sigma = \{x \mapsto fab,y \mapsto ga\}$, ou de façon plus concise $t = c_1[fabga] = c_2[fab]$. Le mot de termes $fabga$ est un suffixe de t.

1.1.3 Graphes

Un *graphe* simple, orienté et étiqueté sur Σ, ou Σ-graphe, est un ensemble $G \subseteq V \times \Sigma \times V$, où Σ est un ensemble fini d'étiquettes et V un ensemble dénombrable quelconque. Un élément (s,a,t) de G est un *arc* de *source* s, d'*extrémité* t et d'*étiquette* a, et se note $s \xrightarrow[G]{a} t$, ou simplement $s \xrightarrow{a} t$ si G peut clairement se déduire du contexte. Un arc dont l'origine et la destination sont identiques est une *boucle*. L'ensemble des sources et extrémités des arcs de G est un sous-ensemble de V appelé support du graphe et noté V. Ses éléments sont appelés *sommets* du graphe. Cette vision des graphes comme des ensembles d'arcs nous permet de définir l'union, l'intersection et la relation d'inclusion sur les graphes comme des opérations ensemblistes classiques. Un graphe inclus dans un autre graphe G est appelé un *sous-graphe* de G. Le sous-graphe de G *induit* par un ensemble de sommets $V' \subseteq V_G$ est composé de l'ensemble des arcs de G dont la source et l'extrémité sont tous deux dans V' (autrement dit $G \cap (V' \times \Sigma \times V')$). On parle aussi parfois de la *restriction* de G à V'.

Une séquence d'arcs $(s_1 \xrightarrow{a_1} t_1, \dots, s_k \xrightarrow{a_k} t_k)$ telle que $\forall i \in [2,k]$, $s_i = t_{i-1}$ est appelée un *chemin*. Il est écrit $s_1 \xrightarrow{u} t_k$, et $u = a_1 \dots a_k$ est son *étiquette de chemin*. Le sommet s_1 est appelé *origine*, et t_k *destination*. Un chemin est un *circuit* si son origine et sa destination sont le même sommet.

Un graphe est *déterministe* (respectivement *co-déterministe*) s'il ne contient aucune paire d'arcs de même source (resp. extrémité) et de même étiquette. Étant donné un sommet v, le nombre d'arcs de source (resp. d'extrémité) v est son *degré sortant* $d^+(v)$ (resp. *degré entrant* $d^-(v)$). La somme de ces deux valeurs est le *degré* de v, $d(v)$. Ces trois mesures sont étendues des sommets aux graphes en considérant le maximum sur tous les sommets, si celui-ci existe. Dans le cas contraire, on dit que le degré est infini.

Une manière équivalente de définir un graphe G est de façon directe comme

un *système de transition étiqueté*, c'est à dire comme un ensemble fini de relations binaires d'arcs $\xrightarrow{a}$ sur V, une pour chaque étiquette $a \in \Sigma$. Le graphe G^{-1} défini comme la famille $(\xrightarrow{a}^{-1})_{a\in\Sigma}$ est le *graphe inverse* de G. Nous notons $\longrightarrow$ la relation d'adjacence définie comme $\bigcup_{a\in\Sigma} \xrightarrow{a}$, $\longleftrightarrow$ sa fermeture par symétrie. Un sous-graphe de G dont le support est fermé par la relation $\xleftrightarrow{*}$ est appelé *composante connexe* de G. Elle est même *fortement connexe* si son support est fermé par $\xrightarrow{*}$. Intuitivement, les composantes connexes d'un graphe sont les parties du graphe dans lesquelles tout sommet est accessible depuis tout autre par un chemin non dirigé (c'est à dire utilisant les arcs dans un sens ou dans l'autre). Un sommet d'un graphe est appelé *racine* s'il existe un chemin depuis ce sommet vers tout autre sommet du graphe. Un graphe qui possède une racine est dit *enraciné*.

Graphes et langages. On peut associer un langage à un graphe en considérant l'ensemble des mots qui étiquettent ses chemins entre deux ensembles déterminés de sommets. Formellement, on appelle *ensemble de traces* (ou *traces*) d'un graphe G à étiquettes dans Σ entre les ensembles de sommets I et F, ou simplement *langage* de G, le langage $L(G,I,F) = \{u \in \Sigma^* \mid \exists s \in I, \exists t \in F, s \xrightarrow[G]{u} t\}$.

Une question importante quant aux traces d'un graphe ou d'une famille de graphes est de bien spécifier quel type d'ensembles de sommets initiaux et finaux est autorisé. Cela induit d'importantes différences par exemple de ne considérer que des ensembles finis plutôt qu'infinis. Nous dirons que deux familles de graphes F_1 et F_2 sont *trace-équivalentes* par rapport à deux familles K_1, K_2 de sous-ensembles de V si, pour tous $G \in F_i$, $I \in K_I$, $F \in K_F$, il existe un graphe $H \in F_{3-i}$ et deux ensembles $I' \in K_I$ et $F' \in K_F$ tels que $L(G,I,F) = L(H,I',F')$.

L'étude des traces de familles de graphes infinies est la motivation principale de plusieurs travaux du domaine des graphes infinis. Nous évoquerons ce point de façon plus détaillée dans le chapitre suivant. Au chapitre 4, nous nous intéresserons tout particulièrement à des familles de graphes dont les traces sont les langages contextuels (cf. § 1.3.3).

Isomorphismes de graphes. L'un des aspects principaux de cette thèse est la comparaison de familles de graphes, de leurs structures, et de certaines de leurs propriétés. Il existe plusieurs façons dont deux graphes peuvent être comparés. Deux graphes sont évidemment identiques s'ils consistent en un même ensemble d'arcs (et donc de sommets), mais l'on s'intéresse en général à des notions d'équivalence moins restrictives. On a déjà cité la notion de trace-équivalence, nous allons à présent rappeler celle d'isomorphisme de graphes.

L'image d'un graphe G de support V_G par une application ϕ de V_G dans un ensemble V est le graphe $\phi(G) = \{(\phi(s),a,\phi(t)) \mid (s,a,t) \in G\}$. Soit H un graphe

de support V, ϕ est un *morphisme de graphes* de G à H si pour tout arc $(s,a,t) \in G$, $(\phi(s),a,\phi(t)) \in H$. Si de plus ϕ est surjectif et ϕ^{-1} est un morphisme surjectif de H à G, ϕ est appelé *isomorphisme* et on dit que G et H sont *isomorphes*. La classe d'équivalence $[G]$ d'un graphe G par isomorphisme est l'ensemble de tous les graphes isomorphes à G. On dit que G est un *représentant* de sa classe d'équivalence. On confond très souvent G et sa classe d'équivalence, et on les note tous les deux simplement G.

Intuitivement, deux graphes sont isomorphes si on peut transformer l'un en l'autre en donnant à chacun de ses sommets un nouveau nom unique. Dans cette thèse, sauf mention expresse du contraire, on considérera toujours les familles de graphes à isomorphisme près, en d'autres termes on s'intéressera toujours de façon implicite à la fermeture par isomorphisme d'une famille plutôt qu'à la famille elle-même. En pratique, pour comparer deux familles de graphes, on oubliera le nommage des sommets en ne les considérant que comme des éléments anonymes d'un ensemble dénombrable arbitraire, pour ne garder que la structure des graphes. Le chapitre suivant approfondira cette idée.

Graphes coloriés. Dans certains cas, il sera utile d'associer une information supplémentaire aux sommets d'un graphe, sous la forme d'étiquettes de sommets, aussi appelées *couleurs*. Étant donné un ensemble fini C de couleurs, un graphe C-colorié est un couple (G,ϕ) où G est un graphe et ϕ une fonction partielle associant à certains sommets de G une couleur dans C. Toutes les définitions précédentes s'étendent simplement à ce nouveau cas, en ignorant simplement les couleurs des sommets. Les couleurs sont le plus souvent utilisées à des fins de modélisation, ou pour la définition de logiques sur les graphes (cf. Section 1.5).

Les termes et les mots vus comme des graphes. Dans un souci d'uniformité, on peut choisir de voir les termes et les mots comme des cas particuliers de graphes coloriés. Les termes dans $T(F,X)$ sont en général représentés par des arbres finis ordonnés dont les nœuds sont étiquetés par les symboles de F ou les variables de X. Les arbres sont des graphes enracinés dans lesquels il existe exactement un chemin depuis la racine vers tout autre sommet. Soit t un terme sur l'alphabet gradué F, on associe à t l'arbre colorié

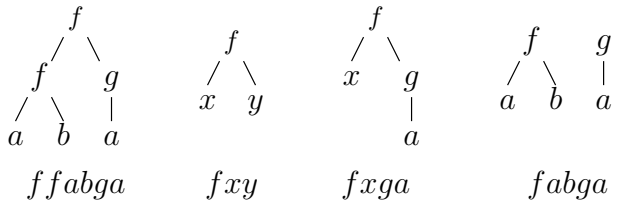
$$T = (\{(u,k,uk) \mid u,v \in Pos(t),k \in \mathbb{N}\},\{u \mapsto f \mid t(u) = f\}).$$

Le support de T est le domaine de t. Chaque sommet de T, appelé un *nœud*, est étiqueté par le symbole fonctionnel à la position correspondante dans t, et les arcs de T relient chaque nœud représentant un symbole fonctionnel à chacun des nœuds représentant ses arguments. Par simplicité, on omet en général les étiquettes et l'orientation des arcs quand on dessine un tel arbre, avec la convention

implicite que les successeurs d'un nœud sont ordonnés de la gauche vers la droite dans l'ordre croissant de leurs étiquettes d'arcs, et en-dessous du nœud lui-même (c'est à dire que la racine se trouve le plus haut, et les feuilles vers le bas).

Dans la suite, on considérera souvent un terme et l'arbre qui le représente comme le même objet, puisqu'ils contiennent tous deux exactement la même information. En particulier, on parlera le plus souvent de langages de termes d'une part et d'automates d'arbres d'autre part, les derniers acceptant les premiers (cf. § 1.3.5.2). Voyant les termes comme des arbres, les mots de termes peuvent être vus comme des forêts ordonnées. Ceci ne devrait pas créer de confusion.

Exemple 1.4. Les termes t, c_1, c_2 et le mot de termes $fabgb$ de l'ex. 1.3 peuvent être représentés par les arbres coloriés suivants:



Nous avons déjà mentionné que les mots peuvent être vus comme des termes sur un alphabet d'arité dont tous les symboles sont d'arité 1 et dont le dernier symbole est une variable. De ce point de vue, ils peuvent aussi être vus comme des arbres de degré sortant borné par 1, ou encore comme des graphes connectés de degré 1.

1.2 Systèmes de réécriture

Les systèmes de réécriture figurent parmi les formalismes les plus généraux de l'informatique pour la modélisation de transformations sur les mots ou les termes[2] (voir par exemple [DJ90]). Ils généralisent les grammaires, peuvent représenter les calculs d'automates finis, de transducteurs, d'automates à pile ou même de machines de Turing, comme nous le verrons plus loin dans ce chapitre. En fait, autant que faire se peut, nous donnerons une caractérisation alternative de chacun de ces formalismes en termes de systèmes de réécriture.

Systèmes de réécriture de mots. Un système de réécriture de mots sur l'alphabet Σ est un ensemble potentiellement infini de règles de réécriture, c'est à dire de couples de mots $(l,r) \in \Sigma^* \times \Sigma^*$. On note $Dom(R)$ (resp. $Im(R)$) l'ensemble des parties gauches et droites de règles de R. Une règle (l,r) *réécrit* un mot w en w' en remplaçant une instance de l dans w par r. Plus formellement, la relation

2. Des notions plus générale de réécriture adaptées aux graphes existent, mais elles ne seront pas abordées ici.

de *réécriture* d'un système R est

$$\underset{R}{\longrightarrow} := \{(ulv,urv) \mid (l,r) \in R \ \wedge \ u,v \in \Sigma^*\}.$$

La clôture réflexive et transitive de cette relation est appelée *dérivation* de R et notée comme d'habitude $\overset{*}{\underset{R}{\longrightarrow}}$. Pour tous u,v tels que $u \overset{*}{\underset{R}{\longrightarrow}} v$, il existe une séquence de mots $u_0 \dots u_n$, appelée *séquence de dérivation*, telle que

$$u = u_0 \underset{R}{\longrightarrow} u_1 \dots u_{n-1} \underset{R}{\longrightarrow} u_n = v.$$

Un mot w ne contenant aucune instance de partie gauche de règle d'un système R est appelé une *forme normale* de R. Un système de réécriture pour lequel tout mot w peut être dérivé en une forme normale est appelé *normalisant*, et *fortement normalisant* si cette forme normale est unique pour chaque mot w donné. L'ensemble des formes normales d'un système R est noté $\mathrm{NF}(R)$.

Systèmes de réécriture de termes. Les systèmes de réécriture de termes peuvent être définis de façon similaire aux systèmes de mots : un système de réécriture de termes sur un alphabet gradué F est composé d'un ensemble potentiellement infini de règles (l,r) où l et r sont des termes de $T(F,X)$ tels que $Var(r) \subseteq Var(l) \subseteq X$ (X est un ensemble de variables supposé disjoint de F). Nous considérons explicitement des ensembles de règles fermés par renommage des variables de X, de façon à ce que les règles qui ne diffèrent que par le nommage de leurs variables puissent être considérées identiques. La relation de réécriture d'un tel système doit être définie par substitution de façon à prendre en compte le fait que les règles de réécriture peuvent modifier la structure des termes, échanger des branches, dupliquer ou supprimer des sous-termes et ainsi de suite. La relation de réécriture d'un système R est [3]

$$\underset{R}{\longrightarrow} := \{(c[l\sigma],c[r\sigma]) \in T(F) \times T(F) \mid (l,r) \in R \ \wedge \ c \in C_1(F) \ \wedge \ \sigma \in S(F,X)\}.$$

Toutes les définitions données précédemment pour les mots s'étendent de façon simple aux termes. Si l'on veut spécifier qu'une règle (l,r) s'applique à la position p (resp. à un ensemble de positions possibles P), on utilise la notation $\underset{l,r}{\longrightarrow}^p$ (resp. $\underset{l,r}{\longrightarrow}^P$). Une règle de réécriture (l,r) est dite *linéaire* si l et r sont tous deux linéaires. Un système ne contenant que des règles linéaires est lui-même appelé linéaire. Un système est *clos* si ses règles ne contiennent aucune variable.

3. Sans perte de généralité, nous ne considérons que la réécriture de termes clos. Les termes contenant des variables peuvent se réécrire en considérant les variables comme des constantes.

Systèmes étiquetés. Dans certains cas il sera utile de considérer des systèmes de réécriture de mots ou de termes dans lesquels chaque règle porte une étiquette d'un alphabet fini Σ. Dans ce nouveau cadre, les règles de réécriture sont des triplets (l,a,r). En plus de sa relation de réécriture $\xrightarrow[R]{}$ définie comme précédemment sans tenir compte des étiquettes, un tel système définit un ensemble fini de relations $(\xrightarrow[R]{a})_{a\in\Sigma}$. Pour tout système étiqueté R, soit R_a le système non étiqueté $R_a = \{(l,r) \mid (l,a,r) \in R\}$. Nous définissons simplement $\xrightarrow[R]{a}$ comme la relation $\xrightarrow[R_a]{}$. Ces relations engendrent un monoïde pour l'opération de composition relationnelle définie par $\xrightarrow[R]{uv} = \{(x,y) \mid \exists z, x \xrightarrow{u} z \xrightarrow{v} y\}$ avec $u,v \in \Sigma^*$. L'élément neutre de ce monoïde est la relation d'identité $\xrightarrow[R]{\varepsilon}$ sur le domaine de réécriture. Une séquence $x_0 \xrightarrow{a_1} x_1 \dots \xrightarrow{a_n} x_n$ est appelée séquence de dérivation étiquetée, son étiquette est le mot $a_1 \dots a_n$. On note toujours $\xrightarrow[R]{*}$ la relation de dérivation non étiquetée de R. On omettra souvent R dans toutes ces notations quand cela ne crée pas d'ambiguïté.

1.3 Une hiérarchie de langages et d'accepteurs

Dans ce paragraphe, nous donnons quelques rappels sur la hiérarchie la plus connue de la théorie des langages formels, à savoir la hiérarchie de langages de Chomsky [Cho59]. Définie il y a plus de 40 ans, elle est formée de quatre des familles des langages les plus étudiées : les langages *récursivement énumérables*, *contextuels*, *hors-contexte* et *réguliers*, qui ont tous des liens avec d'autres problèmes fondamentaux de l'informatique. En particulier, toutes ces familles correspondent à des classes bien connues d'*accepteurs*, parfois appelés *automates* ou *machines*, qui les caractérisent. Ce paragraphe rappelle les quatre classes de grammaires de la hiérarchie de Chomsky utilisées pour définir ces langages, puis détaille chacune des familles d'accepteurs correspondantes.

1.3.1 La hiérarchie de langages de Chomsky

La hiérarchie de Chomsky fut initialement définie en termes de grammaires, qui sont un cas particulier des systèmes de réécriture, dans le but de fournir des outils pour l'étude des langues naturelles. Une grammaire est caractérisée par un quadruplet $G = (T,N,S,P)$, où N et T sont deux alphabets finis disjoints de symboles non-terminaux et terminaux, $S \in N$ est le symbole initial appelé *axiome*, et P est un système de réécriture de mots fini sur l'alphabet $(N \cup T)$, dont les éléments sont en général appelés les règles de la grammaire. Pour plus

de simplicité, on requiert en général que les parties gauches des règles de P contiennent au moins un symbole non-terminal.

On dit qu'un mot $u \in T^*$ est *engendré* par G s'il peut être dérivé depuis l'axiome S selon le système P. Le *langage* de la grammaire G est défini comme l'ensemble de tous les mots terminaux (dans T^*) qui sont engendrés par G :

$$L(G) = \{u \in T^* \mid S \xrightarrow[P]{*} u\}.$$

Remarque 1.5. Par définition des règles de la grammaire, un mot ne contenant que des symboles terminaux ne peut plus être dérivé; il s'agit d'une forme normale pour le système de réécriture P. Le fait de distinguer explicitement les symboles terminaux des non-terminaux n'est pas une caractéristique essentielle des grammaires, qui peuvent en fait être vues simplement comme des systèmes de réécriture.

Les quatre classes de langages de la hiérarchie de Chomsky sont définies en posant des contraintes supplémentaires sur la forme des règles de production des grammaires. Rappelons qu'une première contrainte est que toute partie gauche contienne au moins un non-terminal. Les familles de grammaires considérées sont les suivantes.

- Une grammaire est *non restreinte* ou de *type 0* si aucune contrainte supplémentaire n'est mise sur la forme de ses règles. Ces grammaires engendrent les langages *récursivement énumérables*, aussi appelés langages de type 0.
- Une grammaire est *contextuelle* ou de *type 1* si toutes ses productions sont de la forme (v,w) avec $|w| \geq |v|$. Son langage est appelé *contextuel*, ou de type 1.
- Une grammaire est *hors-contexte* ou de *type 2* si toutes ses productions sont de la forme (A,w) avec $A \in N$ et $w \in (N \cup T)^*$. Le langage d'une telle grammaire est *hors-contexte* ou de type 2.
- Une grammaire est *linéaire à gauche* (ou à droite) si toutes ses règles sont de la forme (A,Bu) ou (A,u) (resp. (A,uB)), où A,B sont des non-terminaux et u est un mot terminal. Une grammaire est dite *régulière* ou de *type 3* si elle est linéaire à gauche ou à droite. Le langage engendré est appelé *régulier* ou de type 3.

Ces quatre familles de grammaires engendrent une hiérarchie stricte de quatre familles de langages, les langages de type 0 étant les plus généraux. Nous verrons dans les paragraphes qui suivent que chacune de ces familles correspond à une famille d'automates ou de machines. Les machines de Turing acceptent les langages de type 0 quand leur mémoire est illimitée, et les langages de type 1 quand l'espace qu'elles peuvent utiliser est au plus linéaire en fonction de la taille de l'entrée. Les langages de type 2 sont acceptés par les automates à pile, et les langages de type 3 par les automates finis.

1.3.2 Machines de Turing

Les machines de Turing sont un modèle proposé par Turing [Tur36] dans les années trente pour tenter de donner une formalisation mathématiques du concept d'algorithme et de calcul automatique. Les machines de Turing consistent intuitivement d'un contrôle fini associé à une mémoire auxiliaire de taille arbitraire sur laquelle peuvent être stockés et lus des symboles.

Les machines de Turing sont très expressives et acceptent l'ensemble de tous les langages de type 0, appelés langages récursivement énumérables. En fait, la célèbre thèse communément appelée « thèse de Church-Turing » avance que les machines de Turing fournissent effectivement une formalisation précise de la notion de calcul algorithmique.

1.3.2.1 Définition

Dans cette âge de l'allocation dynamique de mémoire, nous proposons une définition des machines de Turing différant légèrement de celle que l'on trouve le plus souvent dans les livres de référence (comme par exemple [HU79]).

Dans les définitions classiques, une machine de Turing démarre son calcul avec un mot d'entrée fini inscrit sur une bande de travail infinie divisée en cellules, et travaille en lisant et en inscrivant des symboles sur cette bande. La variante que nous décrivons démarre avec une bande vide et de taille 0, c'est à dire ne contenant aucune cellule, mais a la possibilité d'insérer des nouvelles cellules ou de détruire des cellules existantes à tout moment (en termes de structure de données, pensez à une liste chaînée dynamique plutôt qu'à un tableau). Aussi, les symboles du mot d'entrée sont lus un par un à mesure que le calcul progresse.

Cette notion est inspirée de la notion classique de machines de Turing hors-ligne, aussi appelées machines de Turing étiquetées dans [Cau03].

Définition 1.6 (Machine de Turing)**.** Une machine de Turing est un septuplet $M = (\Sigma,\Gamma, [,],Q,q_0,F,\delta)$, où Σ et Γ sont des ensembles finis de symboles d'*entrée* et de *bande*, $[$ et $]$ $\notin \Gamma$ sont des *délimiteurs de bande*, Q est un ensemble fini d'*états de contrôle*, $q_0 \in Q$ est l'unique *état initial*, $F \subseteq Q$ est un ensemble d'*états finaux* et δ est un ensemble fini de *règles de transition* de l'une des formes suivantes :

$$p[\xrightarrow{l} q[+ \qquad pA \xrightarrow{l} qB\pm \qquad p] \xrightarrow{l} q]- \tag{1.1}$$

$$p[\xrightarrow{l} q[\qquad pA \xrightarrow{l} qB \qquad p] \xrightarrow{l} q] \tag{1.2}$$

$$pA \xrightarrow{l} q \qquad pA \xrightarrow{l} qBA \qquad p] \xrightarrow{l} qB] \tag{1.3}$$

avec $p,q \in Q$, $A,B \in \Gamma$, $\pm \in \{+,-\}$ et $l \in \Sigma \cup \{\varepsilon\}$ [4].

4. Notez que cette définition exclut les règles qui effacent les délimiteurs ou insèrent des cellules hors des limites de la bande.

La bande de travail d'une machine de Turing est composée d'un nombre arbitrairement grand de cellules arrangées selon une ligne, chaque cellule contenant un symbole de Γ. A tout moment, la *configuration* de la machine consiste en son état de contrôle, la taille et le contenu de sa bande, et la position de sa tête de lecture le long de la bande. Nous appellerons cellule courante la cellule qui se trouve en face de la tête de lecture dans la configuration courante.

Chaque type de règles dans la définition ci-dessus décrit un type de transition possible de la machine, accompagnée ou non de la lecture d'un symbole d'entrée selon que $l \in \Sigma$ ou $l = \varepsilon$. Une règle étiquetée par ε est appelée ε-règle, et correspond à une action interne de la machine. La partie gauche d'une règle indique la valeur de l'état de contrôle et le contenu de la cellule courante pour lesquels cette règle est utilisable. La partie droite décrit quelle action effectuer quand la règle est utilisée.

Une règle $pA \xrightarrow{l} qB\pm$ de type (1.1) réécrit le contenu de la cellule courante en B, passe dans l'état de contrôle q et déplace la tête de lecture sur la cellule suivante (vers la droite si $\pm = +$, vers la gauche si $\pm = -$) pourvu que l'état de contrôle soit p et que le contenu de la cellule courante soit A. Notez que quand A est un délimiteur, on doit avoir $B = A$ et le mouvement de la tête est restreint de telle façon que la tête ne peut sortir de la bande. Le type de règles (1.2) décrit la réécriture de la cellule courante sans mouvement de la tête. Une fois encore, les délimiteurs sont un cas particulier et ne peuvent être réécrits. Enfin le type de règles (1.3) décrit l'insertion d'une nouvelle cellule de contenu B à la gauche de la cellule courante, ou la suppression de la cellule courante, pourvu que le contenu de la cellule courante soit A et l'état de contrôle soit p. Quand une nouvelle cellule est ajoutée elle devient la cellule courante, et quand une cellule est supprimée l'ancienne cellule voisine à droite devient la cellule courante. Il faut penser à la suppression d'une cellule comme à la suppression d'un élément dans une liste : même si la cellule est « désallouée », la structure de la liste est conservée en recollant les cellules voisines à droite et à gauche.

1.3.2.2 Configurations et exécutions

Une façon simple et pratique de représenter les configurations de machines de Turing est par des mots de la forme uqv avec $uv \in [\Gamma^*]$, $v \neq \varepsilon$ et $q \in Q$, où uv représente le contenu de la bande (entre délimiteurs), q est l'état de contrôle courant et la tête de lecture pointe sur la cellule qui contient la première lettre de v. Soit C_M l'ensemble des mots de cette forme, ou encore l'ensemble des configurations possibles de M. Il est commode d'interpréter les règles de transition d'une machine de Turing comme un système de réécriture étiqueté sur le domaine C_M. Pour tout $l \in \Sigma \cup \{\varepsilon\}$, on définit la *relation de transition* étiquetée $\xrightarrow[M]{l}$ de M

comme la restriction à $C_M \times C_M$ de la relation de réécriture du système étiqueté

$$\{(pA,Bq) \mid pA \xrightarrow{l} qB+ \in \delta\} \cup \{(pA,qB) \mid pA \xrightarrow{l} qB \in \delta\}$$
$$\cup \{(CpA,qCB) \mid pA \xrightarrow{l} qB- \in \delta \wedge C \in \Gamma \cup \{[\}\}$$
$$\cup \{(pA,q) \mid pA \xrightarrow{l} q \in \delta\} \cup \{(pA,qBA) \mid pA \xrightarrow{l} qBA \in \delta\}.$$

M commence chacun de ses calculs dans sa *configuration initiale* c_0, représentée par le mot $[q_0]$. Ceci correspond à une bande de travail vide, et une tête de lecture dans l'état de contrôle q_0 pointant sur le délimiteur droit. Une configuration est dite *acceptante* ou terminale, si son état de contrôle est dans l'ensemble F. Une séquence $c_0 \xrightarrow[M]{l_1} c_1 \dots c_{n-1} \xrightarrow[M]{l_n} c_n$ est appelée *exécution* (ou calcul) de M sur le mot d'entrée $w = l_1 \dots l_n$. Remarquez qu'on a en général $|w| \leq n$, puisque certains l_i peuvent être le mot vide ε. Une exécution est acceptante ou réussie si elle se termine par une configuration acceptante. Le langage $L(M)$ de M est l'ensemble des mots w pour lesquels il existe au moins une exécution acceptante.

M est dite *déterministe* si elle a au plus un calcul sur tout mot d'entrée. Cette condition est en général indécidable, mais une condition suffisante est qu'aucun couple de règles de transitions ne puisse être utilisé depuis la même configuration à moins qu'elles ne soient étiquetées par des lettres différentes de l'alphabet Σ (et non par ε). On peut montrer que toute machine de Turing peut être transformée en une machine déterministe acceptant le même langage. Une machine est *non-ambiguë* si elle possède au plus un calcul *acceptant* par mot.

Remarque 1.7. Par commodité, on pourra considérer des machines de Turing dont la configuration initiale n'est pas de la forme $[q_0]$ mais est une configuration quelconque c_0 fixée. Ceci n'ajoute pas de pouvoir expressif au modèle, comme on peut facilement le montrer par un encodage de c_0 dans l'ensemble d'états de contrôle de la machine, ce qui ne sera pas détaillé ici. Cette remarque reste valable pour toutes les sous-familles de machines de Turing présentées dans les paragraphes suivants.

1.3.2.3 Langages

Comme mentionné précédemment, les langages acceptés par les machines de Turing sont appelés récursivement énumérables. Ce nom vient précisément du fait que ce sont les ensembles de mots que les machines de Turing peuvent énumérer. Cependant, il est impossible en général de tester de façon effective l'appartenance d'un mot à un tel langage : toute machine de Turing acceptant un langage L peut posséder des calculs infinis non acceptants sur certains mots n'appartenant pas à L. Il est donc impossible de savoir au cours d'un calcul si la machine va finir par s'arrêter et accepter, ou pas. Il est donc intéressant de distinguer une sous-classe stricte de cette famille de langages, appelés langages récursifs, qui sont les

langages des machines de Turing qui s'arrêtent sur tout mot d'entrée. Les langages récursifs jouent un rôle important dans la théorie de la calculabilité. Un problème dont les instances positives peut être encodé comme un langage récursif est dit *décidable*. Une reformulation intuitive de la phrase « le problème P est décidable » est « il existe un algorithme effectif pour résoudre P, c'est à dire un algorithme qui s'arrête sur toute entrée ».

Une propriété bien connue des langages récursivement énumérables est qu'ils ne sont pas clos par complémentaire, mais que tout langage récursivement énumérable dont le complémentaire est aussi récursivement énumérable est en fait récursif. La preuve de l'existence de langages non récursifs (ou non récursivement énumérables) utilise des arguments de diagonalisation (voir par exemple [HU79] pour plus de détails).

Nous allons maintenant donner les définitions de trois autres familles d'accepteurs pour les familles restantes de la hiérarchie de Chomsky. Toutes trois sont définies comme des restrictions syntaxiques des machines de Turing.

1.3.3 Machines linéairement bornées

Lorsqu'elles ne sont autorisées à utiliser qu'un nombre de cellules au plus linéaire en la taille de leur mot d'entrée, les machines de Turing acceptent précisément les langages de type 1, ou langages contextuels. Dans ce cas, elles sont appelées machines linéairement bornées. Cependant, il est indécidable en général de savoir si une machine de Turing donnée est linéairement bornée. Pour cette raison, nous donnons une restriction syntaxique des règles de machines de Turing telle que les machines obtenues soient linéairement bornées et acceptent tous les langages contextuels.

Définition 1.8. Une machine linéairement bornée est une machine de Turing $M = (\Sigma,\Gamma, [,],Q,q_0,F,\delta)$ dont aucune règle d'insertion $pB \xrightarrow{a} qAB \in \delta$ n'est étiquetée par ε.

On peut facilement vérifier que cette restriction implique qu'une telle machine utilise au plus $|w|$ cellules au cours de tout calcul sur le mot w. En effet, à chaque fois que la machine alloue une nouvelle cellule, elle est contrainte de lire un nouveau symbole d'entrée, augmentant ainsi son espace autorisé d'une cellule.

Exemple 1.9. La machine linéairement bornée sur l'alphabet d'entrée et de travail $\{a,b\}$ et l'ensemble d'états de contrôle $\{q_0,q_1,q_2,q_3\}$, avec q_0 l'état initial et q_2 l'unique état final, dont les règles de transition sont

$$q_0] \xrightarrow{a} q_0a] \qquad q_1a \xrightarrow{b} q_1b+ \qquad q_2b \xrightarrow{a} q_3a- \qquad q_3b \xrightarrow{a} q_3a-$$
$$q_0a \xrightarrow{a} q_0aa \qquad q_1] \xrightarrow{\varepsilon} q_2]- \qquad\qquad q_3[\xrightarrow{\varepsilon} q_1[+$$
$$q_0a \xrightarrow{b} q_1b+$$

accepte le langage contextuel $\{(a^n b^n)^+ \mid n \geq 1\}$. Cette machine est déterministe car aucun couple de règles ne peuvent s'appliquer depuis la même configuration. Son unique exécution sur le mot d'entrée *aabbaabb* est

$$[q_0] \xrightarrow{a} [q_0 a] \xrightarrow{a} [q_0 aa] \xrightarrow{b} [bq_1 a] \xrightarrow{b} [bbq_1] \dots$$
$$\dots \xrightarrow{\varepsilon} [bq_2 b] \xrightarrow{a} [q_3 ba] \xrightarrow{a} q_3[aa] \xrightarrow{\varepsilon} [q_1 aa] \xrightarrow{b} [bq_1 a] \xrightarrow{b} [bbq_1] \xrightarrow{\varepsilon} [bq_2 b],$$

après quoi la machine peut soit accepter ce mot (puisque q_2 est final) ou répéter le circuit $[bq_2 b] \xrightarrow{aabb} [bq_2 b]$ un nombre quelconque de fois.

La famille des langages contextuels est strictement incluse dans celle des langages récursifs. Par conséquent, on peut décider si une machine linéairement bornée donnée accepte un mot donné, et aussi si une machine linéairement bornée possède un calcul infini sur un mot d'entrée quelconque. En fait, on peut transformer toute machine linéairement bornée de telle façon que tous ses calculs terminent sur tout mot d'entrée.

Proposition 1.10. *Pour toute machine du Turing linéairement bornée, il existe une machine de Turing linéairement bornée terminante acceptant le même langage.*

Démonstration. Il suffit de montrer que pour toute machine linéairement bornée M dont l'ensemble de transitions contient un circuit, il existe une machine terminante équivalente M', c'est à dire une machine sans calcul infini acceptant le même langage que M. Le nombre total de configurations distinctes de M pendant un calcul sur un mot de longueur n est borné par k^n, où k est une constante dépendant de la taille de l'alphabet de travail et de l'ensemble d'états de contrôle de M. Il est facile de voir que tout mot w accepté par M doit être accepté par au moins un calcul de taille inférieure à $k^{|w|}$. En effet, si le calcul le plus court sur w était de longueur supérieure à cette borne, il contiendrait nécessairement au moins deux occurrences distinctes de la même configuration, c'est à dire un circuit. En retirant ce circuit, on obtient un calcul acceptant plus court.

Soit M' la machine linéairement bornée qui simule M tout en incrémentant un compteur en base k à n chiffres encodé dans l'alphabet (un chiffre par cellule). A chaque fois qu'une cellule est insérée, le compteur est remis à 0, mais dispose d'un chiffre supplémentaire. Le calcul en cours de simulation est arrêté si le compteur déborde. Par construction, M' accepte le même langage que M mais ne possède de calcul infini sur aucun mot d'entrée. Des arguments semblables ont été utilisés dans [Kur64] pour montrer que l'ensemble des mots sur lesquels une machine linéairement bornée a un calcul infini est contextuel. □

A la différence des langages récursivement énumérables, les langages contextuels forment une algèbre de Boole : ils sont clos par union, intersection et complémentaire.

Théoreme 1.11 ([Imm88]). *Les langages contextuels sur un alphabet* Γ *forment une algèbre de Boole effective.*

La notion de machine de Turing bornée en espace peut être étendue à une borne $f : \mathbb{N} \mapsto \mathbb{N}$ quelconque telle que pour tout $n \geq 0$, $f(n) \geq n$. L'ensemble des langages acceptés par une machine de Turing en espace $f(n)$ sur une entrée de taille n est noté NSPACE$[f(n)]$. Le théorème suivant énonce la propriété que la hiérarchie spatiale est stricte.

Théoreme 1.12 ([HU79, Imm88]). *Pour toute paire de fonctions espace-constructibles* f *et* g *de* $\mathbb{N} \mapsto \mathbb{N}$ *telles que* $\lim_{n\to+\infty} f(n)/g(n) = 0$*, on a l'inclusion stricte* NSPACE$[f(n)] \subset$ NSPACE$[g(n)]$.

En particulier, la famille NSPACE$[2^n]$ des langages reconnaissables en espace exponentiel contient strictement la famille des langages contextuels (NSPACE$[n]$).

Comme ils sont acceptés par une restriction syntaxique des machines de Turing, les langages contextuels sont récursivement énumérables (et même récursifs, grâce à leur fermeture par complémentaire par exemple). L'existence de langages récursifs non contextuels fait appel à des arguments de diagonalisation. Contrairement aux langages récursivement énumérables, on ne sait pas à l'heure actuelle si tout langage contextuel peut être accepté par une machine linéairement bornée déterministe. Cette question a été posée par Kuroda dans [Kur64].

1.3.4 Automates à pile

Au niveau suivant de la hiérarchie de Chomsky se trouvent les langages hors-contexte. Cette famille est aussi caractérisée par la classe bien connue des automates à pile, qui en plus de leur contrôle fini utilisent une mémoire auxiliaire organisée comme une pile. À chaque transition, ils peuvent consulter le symbole de sommet de pile, et soit le défausser soit le remplacer par d'autres symboles. Comme nous l'avons fait pour les machines linéairement bornées, nous présentons les automates à pile comme une restriction syntaxique des machines de Turing. La pile sera représentée par le contenu de la bande de travail, avec la convention que les cellules les plus à gauche représentent les éléments les plus haut dans la pile.

Définition 1.13. Un automate à pile est une machine de Turing $P = (\Sigma,\Gamma,[,],Q,q_0,F,\delta)$ ne possédant aucune règle de type (1.1) (cf. définition 1.6).

Cette restriction assure que la tête de lecture d'un automate à pile ne se déplace jamais le long de la bande, et fait toujours face au second symbole en partant de la gauche (le premier étant le délimiteur gauche). Toutes les configurations accessibles par P depuis la configuration initiale $[q_0]$ sont en effet de la forme $[qs]$, où $q \in Q$ et $s \in \Gamma^*$, que nous abrégeons en qs. Lorsque l'on travaille avec des automates à pile, la bande et son contenu sont appelés *la pile*. Les règles

d'insertion sont vues comme l'addition d'un nouveau symbole en sommet de pile (empilage), opération habituellement appelée *push*, et les règles de suppression comme le fait de retirer un symbole en sommet de pile (opération de dépilage ou *pop*). Les règles de réécriture de la cellule courante sont équivalentes à un *pop* suivi d'un *push*.

Exemple 1.14. L'automate à pile sur l'alphabet d'entrée $\{a,b\}$, l'alphabet de pile $\{a\}$ et les états de contrôle $\{q_0,q_1,q_2\}$, avec q_0 initial et q_2 final, et dont les transitions sont

$$q_0] \xrightarrow{a} q_0a] \qquad q_0a \xrightarrow{b} q_1 \qquad q_1] \xrightarrow{\varepsilon} q_2]$$
$$q_0a \xrightarrow{a} q_0aa \qquad q_1a \xrightarrow{b} q_1$$

accepte le langage hors-contexte $\{a^nb^n \mid n \geq 1\}$. Cet automate à pile est déterministe car aucun couple de règles ne peut s'appliquer depuis la même configuration. Son unique exécution sur le mot d'entrée a^nb^n est acceptante, c'est la séquence

$$[q_0] \xrightarrow{a} [q_0a] \xrightarrow{a} [q_0aa] \xrightarrow{a} \dots$$
$$\dots \xrightarrow{a} [q_0a^n] \xrightarrow{b} [q_1a^{n-1}] \xrightarrow{b} [q_1a^{n-2}] \xrightarrow{b} \dots$$
$$\dots \xrightarrow{b} [q_1] \xrightarrow{\varepsilon} [q_2].$$

Dans certaines définitions des automates à pile, des règles générales de la forme $pA \xrightarrow{a} qW$ avec $W \in \Gamma^*$ sont autorisées. De telles règles permettent, en une seule étape, de dépiler le symbole de sommet de pile A et d'empiler successivement toutes les lettres de W en commençant par la dernière (de sorte que la première lettre de W devient le nouveau sommet de pile). Il est aussi parfois requis que seules les configurations où la pile est vide soient acceptantes.

Ces différences de définition ne changent en fait pas l'expressivité du modèle, et de simples constructions permettent de simuler ces différents formalismes à l'aide de celui que nous avons présenté, moyennant éventuellement une augmentation de la taille de l'alphabet de pile. A la différence des langages contextuels, les langages hors-contexte ne sont clos ni par complémentaire ni par intersection, et contrairement aux machines de Turing les automates à pile ne peuvent pas toujours être déterminisés : il existe des langages hors-contexte qui ne sont acceptés que par des automates à pile non déterministes.

Les langages hors-contexte sont strictement inclus dans les langages contextuels. Par exemple, les langages $\{a^nb^nc^n \mid n \in \mathbb{N}\}$ ou le langage $\{(a^nb^n)^+ \mid n \geq 1\}$ de l'exemple 1.9 sont contextuels mais pas hors-contexte.

Automates à pile d'ordre supérieur

Entre la classe des langages hors-contexte et celle des langages contextuels a été définie une hiérarchie stricte de langages appelée hiérarchie OI [Dam82]. Ils

sont caractérisés comme les langages acceptés par des *automates à pile d'ordre supérieur*. En plus de manipuler les symboles de piles eux-mêmes, ces automates sont capables de dupliquer ou de détruire des piles entières de symboles, des piles de piles, et ainsi de suite. Avant de présenter la définition formelle des automates à pile d'ordre supérieur, nous définissons les piles d'ordre supérieur et les opérations associées.

Une *pile de niveau 1* sur l'alphabet Γ est un mot $w \in \Gamma^*$, noté $[w]$. La pile vide de niveau 1 est notée $[\,]$. Pour $n \geq 2$, une *pile de niveau n* sur Γ est une séquence non vide $[s_1 \dots s_k]$ de piles de niveau $n-1$. L'ensemble des piles de niveau $n \geq 1$ est noté S_n, et le niveau d'une pile s donnée $l(s)$. Les opérations suivantes sont définies sur les piles de niveau 1 :

$$\begin{aligned} push_1^a([a_1 \dots a_l]) &= [a a_1 a_2 \dots a_l] && \text{pour tout } a \in \Gamma, \\ pop_1([a_1 \dots a_l]) &= [a_2 \dots a_l] && \text{pour } l \geq 1, \end{aligned}$$

Les opérations suivantes sont définies sur les piles de niveau $n > 1$:

$$\begin{aligned} push_1^a([s_1 \dots s_l]) &= [push_1^a(s_1) \dots s_l] && \text{pour tout } a \in \Gamma, \\ push_k([s_1 \dots s_l]) &= [push_k(s_1) \dots s_l] && \text{pour } k \in [2,n[, \\ push_n([s_1 \dots s_l]) &= [s_1 s_1 \dots s_l] \\ pop_k([s_1 \dots s_l]) &= [pop_k(s_1) \dots s_l] && \text{pour } k \in [1,n[, \\ pop_n([s_1 \dots s_l]) &= [s_2 \dots s_l] && \text{pour } l > 1, \text{ sinon non défini.} \end{aligned}$$

Nous définissons aussi une fonction *top* permettant d'inspecter le symbole de sommet de la pile de niveau 1 la plus haute. On a $top_1([a_1 \dots a_l]) = a_1$ pour $[a_1 \dots a_l] \in S_1$, $top_1([\,]) = \varepsilon$ et $top([s_1 \dots s_l]) = top(s_1)$ pour $[s_1 \dots s_l]$ de niveau supérieur à 1. On note O_n l'ensemble des opérations définies sur les piles de niveau n· On dit qu'une opération o est de niveau n, et on note $\mathrm{l}(o) = n$, si o est $push_n$ ou pop_n, ou $push_1^a$,pop_1 si $n = 1$. On peut à présent définir les automates à pile de niveau supérieur.

Définition 1.15. Un automate à pile de niveau n est un octuplet $P = (\Sigma,\Gamma, [,],Q,q_0,F,\delta)$, où Σ et Γ sont l'alphabet d'entrée et l'alphabet de pile, [et] $\notin \Gamma$ sont des délimiteurs de pile, Q est un ensemble fini d'états de contrôle, $q_0 \in Q$ est l'unique état initial, $F \subseteq Q$ est un ensemble d'états finaux et δ est un ensemble fini de règles de transition de la forme $pA \xrightarrow{l} qo$ avec $p,q \in Q$, $A \in \Gamma \cup \{\varepsilon\}$, $o \in O_n$ et $l \in \Sigma \cup \{\varepsilon\}$.

Le niveau d'une règle de transition $pA \xrightarrow{l} qo$ est simplement $\mathrm{l}(o)$. Les configurations d'un automate à pile P de niveau n sont des couples (q,s), où $q \in Q$ et s est une pile de niveau n sur Γ. Les règles de transition de P induisent une relation de transition étiquetée $(\xrightarrow[P]{l})_{l \in \Sigma \cup \{\varepsilon\}}$ sur son ensemble de configurations telle que

$(p,s) \xrightarrow[P]{l} (q,s')$ si et seulement si $top(s) = A$, P possède une règle $pA \xrightarrow{l} qo$ et $s' = o(s)$. Les calculs, le déterminisme et la notion de langage accepté sont définis comme d'habitude.

Exemple 1.16. L'automate à pile de niveau 2 sur l'alphabet d'entrée $\{a,b,c\}$ et l'alphabet de pile $\{a\}$, d'états de contrôle $\{q_0,q_1,q_2,q_3\}$ avec q_0 initial et q_3 final et dont les transitions sont

$$\begin{array}{lll} q_0\varepsilon \xrightarrow{a} q_0 push_1^a & q_1 a \xrightarrow{b} q_1 pop_1 & q_2 a \xrightarrow{c} q_2 pop_1 \\ q_0 a \xrightarrow{a} q_0 push_1^a & q_1\varepsilon \xrightarrow{\varepsilon} q_2 pop_2 & q_2\varepsilon \xrightarrow{\varepsilon} q_3 push_2 \\ q_0 a \xrightarrow{\varepsilon} q_1 push_2 & & \end{array}$$

accepte le langage $\{a^n b^n c^n \mid n \geq 1\}$. Son unique exécution sur le mot d'entrée $a^n b^n c^n$ est la séquence

$$\begin{aligned} (q_0,[[\,]]) &\xrightarrow{a} (q_0,[[a]]) \xrightarrow{a} (q_0,[[aa]]) \xrightarrow{a} \dots \\ &\dots \xrightarrow{a} (q_0,[[a^n]]) \xrightarrow{\varepsilon} (q_1,[[a^n][a^n]]) \xrightarrow{b} (q_1,[[a^{n-1}][a^n]]) \xrightarrow{b} \dots \\ &\dots \xrightarrow{b} (q_1,[[\,][a^n]]) \xrightarrow{\varepsilon} (q_2,[[a^n]]) \xrightarrow{c} (q_2,[[a^{n-1}]]) \xrightarrow{c} \dots \\ &\dots \xrightarrow{c} (q_2,[[a]]) \xrightarrow{c} (q_2,[[\,]]) \xrightarrow{\varepsilon} (q_3,[[\,][\,]]). \end{aligned}$$

Les automates de niveau 1 sont bien sûr identiques aux automates à pile classiques, et acceptent les langages hors-contexte. Les langages acceptés à chaque niveau forment une hiérarchie stricte et infinie, mais sont tous contextuels. Les langages $\{ww \mid w \in \Sigma^*\}$ et $\{a^n b^n c^n\}$ sont des exemples de langages non hors-contexte acceptés par des automates de niveau 2. Pour plus de détails sur ces automates et cette hiérarchie de langages, voir par exemple [Dam82, Eng83].

1.3.5 Automates finis et notions associées

Nous présentons maintenant trois des caractérisations les plus courantes des langages de type 3, ou langages réguliers, et en particulier les automates finis. L'étude des automates finis et de leurs variantes est un sujet très riche possédant des liens avec de nombreux autres domaines de l'informatique, depuis la vérification de programmes jusqu'à la théorie des semigroupes. Pour une introduction aux bases de la théorie, le lecteur intéressé est invité à consulter [HU79, Sak03]. Les applications possibles des automates finis sont trop nombreuses pour être énumérées. Le paragraphe 1.6 mentionne certains aspects de la vérification de programmes utilisant la théorie des automates.

Les automates finis, à l'instar des machines linéairement bornées et des automates à pile, peuvent être définis comme une restriction simple des machines de

Turing telles que nous les avons définies. En fait, les automates finis n'utilisent que la mémoire finie implicitement assurée par leur ensemble d'états de contrôle, mais ne font aucun usage de la bande de travail.

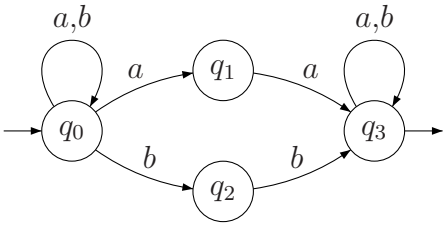
Définition 1.17. Un automate fini $A = (\Sigma,\Gamma,[,], Q,q_0,F,\delta)$ est une machine de Turing avec uniquement des règles de la forme $p] \xrightarrow{l} q]$, avec $p,q \in Q$ et $l \in \Sigma \cup \{\varepsilon\}$.

En raison de cette restriction, les configurations accessibles depuis $[q_0]$ dans un automate fini sont toutes de la forme $[q]$, ou plus simplement q, où q est un état de contrôle. Les automates finis doivent évidemment leur nom au fait que le nombre de telles configurations est fini. Une règle de transition $p] \xrightarrow{a} q]$ sera également abrégée en $p \xrightarrow{a} q$. Comme les automates finis n'utilisent pas du tout la bande de travail, on omettra en général Γ,[et] et on notera simplement $A = (\Sigma,Q,q_0,F,\delta)$. Les automates finis sont le plus couramment représentés par des diagrammes d'états-transitions, c'est à dire par des graphes dont les sommets représentent les états de contrôle d'un automate et dont les arcs représentent les transitions. Les états initiaux et finaux sont marqués avec une flèche entrante ou sortante, qui peuvent être vues comme des couleurs.

Exemple 1.18. L'automate $A = (\{a,b\},\{q_0,q_1,q_2,q_3\},q_0,q_1,\delta)$ possédant les transitions

$$\delta = \{q_0 \xrightarrow{a} q_0,\ q_0 \xrightarrow{b} q_0,\ q_0 \xrightarrow{a} q_1,\ q_0 \xrightarrow{b} q_2,\\ q_1 \xrightarrow{a} q_3,\ q_2 \xrightarrow{b} q_2,\ q_3 \xrightarrow{a} q_3,\ q_3 \xrightarrow{b} q_3\}$$

est représenté par le diagramme

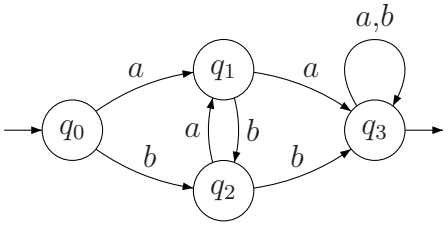


Le langage $L(A)$ de cet automate est l'ensemble des mots sur $\{a,b\}$ contenant au moins une occurrence de l'un des facteurs aa ou bb.

La classe de langages acceptés par les automates finis est précisément celle des langages de type 3 de la hiérarchie de Chomsky, en général appelés langages réguliers. À la différence des automates à pile, les automates finis peuvent être transformés en un automate (minimal) déterministe reconnaissant le même langage et ne possédant pas d'ε-transitions.

Exemple 1.19. L'automate déterministe représenté par le diagramme suivant accepte le même langage que l'automate de l'exemple 1.18. Il est l'automate

déterministe minimal associé à ce langage.



Les langages réguliers sont clos par de nombreuses opérations, dont l'union et le complémentaire (et donc également l'intersection). Ils sont strictement inclus dans la famille des langages hors-contexte (par exemple, le langage $\{a^n b^n \mid n \in \mathbb{N}\}$ est hors-contexte mais pas régulier).

Les automates finis et leurs langages sont l'une des familles d'objets les plus fondamentales de l'informatique théorique, et même de l'informatique en général. Plusieurs autres caractérisations des langages réguliers existent, en particulier comme les parties rationnelles ou reconnaissables du monoïde libre Σ^*.

1.3.5.1 Parties rationnelles et reconnaissables d'un monoïde

La reconnaissabilité et la rationalité sont des notions algébriques définies sur les monoïdes. La première notion utilise la notion de morphisme de monoïdes, et la seconde utilise l'opération dite « étoile de Kleene » sur les monoïdes.

Un (homo)morphisme d'un monoïde M sur un monoïde N est défini comme une application ϕ de M sur N telle que $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$ pour tous a,$b \in M$, et $\phi(1_M) = 1_N$. Par conséquent, pour tout ensemble S de générateurs de M, ϕ est entièrement définie par sa restriction à S. Dans le cas particulier où M est le monoïde libre Σ^*, il est donc suffisant de spécifier $\phi(a)$ pour tout $a \in \Sigma$ pour définir ϕ. Une partie *reconnaissable* R d'un monoïde M est une partie de M pour laquelle il existe un morphisme *d'image finie* ϕ tel que $\phi^{-1}(\phi(R)) = R$.

Nous définissons maintenant l'étoile de Kleene (ou étoile) sur un monoïde M. Soit S une partie quelconque de M, on définit S^0 comme le singleton $\{\varepsilon\}$, et S^i récursivement comme $S \cdot S^{i-1}$ pour tout $i > 0$, où $\cdot$ est l'opération de produit associée à M et étendue aux ensembles. L'étoile de S est définie comme $S^* = \bigcup_{i \geq 0} S^i$. On écrit aussi parfois S^+ l'ensemble $\bigcup_{i>0} S^i$. L'ensemble des parties *rationnelles* de M est le plus petit ensemble contenant les parties finies de M qui soit fermé par union, produit et étoile. En d'autres termes, une partie de M est rationnelle si elle peut s'exprimer à l'aide des opérateurs union, produit et étoile et des parties finies de M. Les parties rationnelles du monoïde libre sont donc très souvent décrites à l'aide d'expressions sur les symboles de l'alphabet et sur ces opérateurs, appelées expressions rationnelles.

Un résultat important de l'informatique est le fait qu'à la fois les ensembles rationnels et les ensembles reconnaissables du monoïde libre coïncident avec les

langages reconnus par les automates finis [Kle56] et avec les langages définis en logique du second ordre monadique [Büc62] (cf. § 1.5.2), et que des algorithmes efficaces existent pour effectuer le passage de chacun de ces formalismes vers tous les autres. Se référer à [Wei04] pour un résumé clair et concis de ces résultats, et à [Sak03] par exemple pour une présentation plus détaillée. L'équivalence entre les parties rationnelles et reconnaissables n'est pas vérifiée pour tous les monoïdes de type fini. Dans le cas général, c'est à dire pour les monoïdes de type fini non libres; les parties reconnaissables sont aussi rationnelles mais la réciproque est fausse [Kni64]. C'est par exemple le cas du monoïde des relations binaires sur les mots (voir § 1.4).

1.3.5.2 Automates finis d'arbres

Après le développement de la théorie des langages de mots, des notions de langages et d'automates généralisées au cadre des termes et arbres finis sont apparues (voir [CDG+] pour une introduction au domaine). Malgré de récents progrès [EW05], la connaissance des structures algébriques sous-tendant les termes et les arbres est beaucoup moins développée que celle du monoïde libre, et l'on ne peut pas réellement parler à l'heure actuelle d'une véritable théorie algébrique des langages de termes. Cependant, quelques notions ont été adoptées dans la communauté comme les extensions les plus naturelles du cas de mots, en particulier en ce qui concerne les caractérisations équivalentes par logique, par automates, par des expressions rationnelles généralisées ou par une notion algébrique de reconnaissabilité des ensembles *réguliers* de termes. Nous ne présenterons pas en détail l'ensemble des éléments de cette équivalence, mais nous concentrerons sur le cas des automates d'arbres, qui seront utilisés à plusieurs reprises dans ce document.

Les automates d'arbres finis acceptent en entrée des arbres finis coloriés représentant des termes sur un alphabet gradué F[5]. Ils sont semblables aux automates de mots en ceci qu'ils évaluent leur entrée symbole par symbole tout en maintenant une mémoire finie grâce à leur ensemble d'états de contrôle. Cependant, en raison de la topologie plus complexe des arbres, les automates d'arbres finis doivent poursuivre simultanément leurs calculs sur plusieurs branches de l'entrée simultanément. Ceci peut se faire principalement de deux façons : soit l'arbre est parcouru en commençant par les feuilles et en progressant vers la racine (parcours « ascendant » ou *bottom-up*), soit dans l'autre sens (parcours « descendant » ou *top-down*).

5. Rappelons ici que la notion d*arbre ordonné colorié représentant un terme* est indissociable de la notion de *terme*. Nous utiliserons sans distinction les mots *arbre* et *terme*, ainsi que le vocabulaire associé (branche, sous-terme, feuille, etc.) quand il sera question d'automates d'arbres.

Automates ascendants.

Définition 1.20. Un automate d'arbres fini ascendant est un quadruplet $A = (F,Q,Q_f,\delta)$, où F est un alphabet gradué d'entrée, Q un ensemble fini d'états de contrôle, $Q_f \subseteq Q$ un ensemble d'états finaux et δ un ensemble de règles de transition de la forme $f(q_1,\ldots,q_n) \to q$.

Pendant leurs calculs, les automates finis sur les mots associent intuitivement un état de contrôle à chaque préfixe de leur mot d'entrée, et plus précisément l'état qu'ils atteignent une fois ce préfixe lu, partant de l'état initial. De façon analogue, les automates d'arbres ascendants associent un état de contrôle à chaque sous-terme clos lu en entrée. Une règle $f(q_1,\ldots,q_n) \to q$ d'un automate ascendant A stipule qu'un terme $ft_1\ldots t_n$ tel que chaque t_i est associé par A à l'état de contrôle q_i doit lui-même être associé à l'état de contrôle q[6]. Plus formellement, la relation de transition $\underset{A}{\longrightarrow}$ de A est définie comme la relation de dérivation du système de réécriture de termes

$$\{(fq_1x_1\ldots q_nx_n,qfx_1\ldots x_n) \mid f(q_1,\ldots,q_n) \to q \in \delta\}$$

sur $T(F \cup Q)$, où les éléments de Q sont considérés comme des symboles unaires. Un calcul de A sur le terme d'entrée t est une séquence de dérivation de ce système d'origine t. Le terme t est accepté par A s'il peut être dérivé en q_ft, pour un certain état final $q_f \in Q_f$. Comme d'habitude, A est dit déterministe s'il admet au plus un calcul sur tout terme d'entrée (ou, de façon équivalente, s'il possède au plus une règle ayant la même partie gauche), et non-ambigu s'il possède au plus un calcul *acceptant* par terme d'entrée.

Exemple 1.21. L'automate d'arbres ascendant déterministe sur l'alphabet $\{f^{(2)}, g^{(1)},a^{(0)},b^{(0)}\}$ ayant pour état final q_2 et pour règles de transition

$$\begin{array}{lll} a \to q_0 & gq_0 \to q_0 & fq_0q_1 \to q_2 \\ b \to q_1 & gq_1 \to q_1 & fq_1q_0 \to q_2 \end{array}$$

accepte l'ensemble de termes $\{f(g^*a,g^*b)\} \cup \{f(g^*b,g^*a)\}$. L'unique calcul sur le terme d'entrée $f(gga,gb)$ peut être décrit par la séquence de dérivation

$$\begin{array}{c} f(gga,gb) \longrightarrow f(gg(q_0a),gb) \longrightarrow f(g(q_0ga),g(q_1b)) \\ \longrightarrow f(q_0gga,q_1gb) \longrightarrow q_2(f(gga,gb)). \end{array}$$

Automates descendants.

Définition 1.22. Un automate d'arbre fini descendant est un quadruplet $A = (F,Q,Q_i,\delta)$, où F est un alphabet gradué d'entrée, Q un ensemble fini d'états

6. Notez que dans le cas particulier où f est une constante, disons a, la forme de la règle devient $a \to q$, signifiant que le terme consistant d'une unique feuille étiquetée par a peut être associé à l'état q.

de contrôle, $Q_i \subseteq Q$ un ensemble d'états initiaux et δ un ensemble de règles de transition de la forme $q \rightarrow f(q_1, \ldots, q_n)$.

La sémantique des automates d'arbres descendants est précisément duale de celle des automates ascendants, et la relation de transition de A est définie comme l'inverse de l'automate ascendant correspondant $A' = (F, Q, Q_i, \delta^{-1})$ dont l'ensemble d'états finaux est Q_i et dont les règles sont les règles de A dont le sens des flèches est inversé. Par conséquent, tout calcul de A est la fin d'un calcul inversé de A', et un calcul acceptant de A est un calcul acceptant inversé de A'. Par conséquent, les automates ascendants et descendants ont évidemment le même pouvoir expressif et reconnaissent les mêmes ensembles de termes, que nous appellerons dorénavant ensembles réguliers de termes. Cette famille est close par les opérations booléennes classiques et par homomorphismes linéaires et homomorphismes inverses.

La seule différence entre ces deux variantes d'automates d'arbres réside dans la notion de déterminisme. En général, l'automate descendant correspondant à un automate ascendant déterministe n'est pas déterministe. Les automates ascendants déterministes et non-déterministes sont aussi expressifs, mais ce n'est pas le cas des automates descendants, dont la version déterministe est strictement moins expressive. Le langage accepté par l'automate de l'exemple 1.21 est accepté par un automate descendant non-déterministe mais par aucun automate descendant déterministe.

Exécutions partielles. Nous introduisons une terminologie spéciale relative aux calculs des automates d'arbres sur des mots de termes. On dit qu'un mot de terme clos t est *accepté* par un automate descendant A si chacun des termes le composant est accepté par A On dit qu'un n-contexte t est *partiellement accepté* par A depuis l'état de contrôle q jusqu'au mot de contrôle $q_1 \ldots q_n$ s'il existe un calcul de l'automate depuis la configuration $q(t)$ jusqu'à la configuration $t[q_1(\Box_1) \ldots q_n(\Box_n)]$. Un mot de termes k'-contextuel $t_1 \ldots t_k$ de longueur k est partiellement accepté par A depuis le mot de contrôle $q_1 \ldots q_k$ jusqu'au mot de contrôle $q'_1 \ldots q'_{k'}$ si chaque t_i est partiellement accepté par A depuis l'état q_i jusqu'au mot de contrôle u_i et $u_1 \ldots u_k = q'_1 \ldots q'_{k'}$. Les mots $q_1 \ldots q_k$ et $q'_1 \ldots q'_{k'}$ sont appelés respectivement mot de contrôle initial et final de ce calcul. Cette terminologie s'étend par dualité aux automates ascendants.

1.3.6 Quelques remarques de plus

Les quatre familles d'accepteurs de mots que nous venons de présenter forment effectivement une hiérarchie, comme on peut le constater par les définitions successives des machines linéairement bornées, des automates à pile et des automates

finis par restrictions successives des machines de Turing. Les langages qu'ils acceptent sont les langages de la hiérarchie de Chomsky, qui est stricte.

Des machines et des automates sur des monoïdes plus généraux (en particulier sur des monoïdes non libres) peuvent être définis en remplaçant l'alphabet d'entrée Σ dans les définitions précédentes par tout sous-ensemble fini d'un monoïde quelconque, et l'étiquette ε par l'élément neutre de ce monoïde. Le long d'un calcul, on ne considérera plus la concaténation des symboles d'entrée mais plutôt leur produit dans le monoïde pour définir l'ensemble des éléments acceptés par la machine. Un exemple d'automates finis sur un monoïde non libre est le cas des transducteurs finis, présentés au paragraphe 1.4.2.

Pour finir ce paragraphe, nous voudrions insister sur le fait que chacune de ces quatre familles d'accepteurs finis peut être vue comme une famille de systèmes de réécriture de mots sur un certain domaine. Ceci n'est guère surprenant étant donné que nous avons utilisé des systèmes de réécriture pour définir les relations de transition des machines de Turing, mais garder ce fait à l'esprit aura de l'importance vis-à-vis de certains des propos exposés ici (voir en particulier le paragraphe 2.1.1).

1.4 Familles de relations binaires

Ce paragraphe présente certaines des familles de relations binaires sur les mots les plus connues. Nous considérons des ensembles de paires de mots vues comme des éléments du monoïde produit $\Sigma^* \times \Sigma^*$, dont la loi de composition est définie par $(u_1,v_1) \cdot (u_2,v_2) = (u_1u_2,v_1v_2)$. Il est naturel de considérer les parties rationnelles et reconnaissables de ce monoïde, que nous appellerons respectivement relations rationnelles et relations reconnaissables de mots. Notez que $\Sigma^* \times \Sigma^*$ est de type fini, car engendré par l'ensemble fini $(\Sigma \times \{\varepsilon\}) \cup (\{\varepsilon\} \times \Sigma)$, mais n'est pas libre, et les familles d'ensembles rationnelles et reconnaissables ne coïncident plus comme dans le cas des mots. Nous présentons ces deux familles à l'aide d'automates, ainsi que quelques autres familles intermédiaires de relations. Pour conclure, nous évoquons des extensions possibles de chacune de ces notions au cas des termes.

1.4.1 Relations reconnaissables (et quelques extensions)

Les parties reconnaissables d'un produit direct de deux monoïdes ont la propriété intéressante suivante, habituellement appelée théorème de Mezei : étant donnés deux monoïdes M et N, les parties reconnaissables du monoïde produit $M \times N$ peuvent être vues comme des unions finies de produits $U_i \times V_i$, où chaque U_i et V_i est une partie reconnaissable du monoïde correspondant. Dans le cas des relations binaires sur les mots, ceci signifie que chaque relation reconnaissable sur

$\Sigma^* \times \Sigma^*$ peut être vue comme une union finie de produits $K_i \times L_i$, où tous les K_i et L_i sont des langages réguliers.

Partant de ce résultat, il est possible de caractériser toute relation reconnaissable R sur les mots de Σ^* par un ensemble fini de paires d'automates finis (A_i,B_i) sur Σ. On dit alors qu'une paire de mots (u,v) appartient à R si $u \in L(A_i)$ et $v \in L(B_i)$ pour au moins un i.

Exemple 1.23. La relation binaire de $\{a\}^*$ dans $\{b\}^*$ qui à tout mot de longueur paire associe un mot quelconque de longueur impaire (et réciproquement) est reconnaissable car elle peut être écrite $\big((a^2)^* \times (b^2)^*b\big) \cup \big((a^2)^*a \times (b^2)^*\big)$.

Relations préfixe-reconnaissables

Les relations reconnaissables sont plutôt faibles, et ne permettent aucune forme de synchronisation entre un mot et son image. Par exemple, la relation identité n'est pas reconnaissable. Une légère extension de ces relations permettant de spécifier des relations plus intéressantes est de considérer les *clôtures rationnelles* (à droite ou à gauche) de relations reconnaissables [Cho82]. Une relation R' est la clôture rationnelle à droite d'une relation reconnaissable $R = \bigcup_{i\in[1,n]}(U_i \times V_i)$ si elle consiste en un ensemble fini de relations $\{(uw,vw) \mid (u,v) \in U_i \times V_i, w \in W_i\}$, où chaque U_i,V_i et W_i est un langage régulier (ou rationnel). On qualifie parfois ces relations de *préfixe-reconnaissables* [Cau96, CC03]. L'ensemble des relations préfixe-reconnaissables forme une algèbre de Boole, et il est fermé par composition et fermeture transitive de la composition.

Un cas particulier de fermeture rationnelle est la fermeture par le langage Σ^*, que nous appellerons simplement *clôture* à droite ou à gauche de relations reconnaissables. Cette classe à des propriétés moins intéressantes que celle des relations préfixe-reconnaissables puisqu'elle n'est close ni par complémentaire ni par intersection.

Exemple 1.24. La relation identité sur l'alphabet Σ est la fermeture par Σ^* de la relation reconnaissable $\{\varepsilon\} \times \{\varepsilon\}$.
La relation $\{(a^n,b^n) \mid n \geq 0\}$ de $\{a\}^*$ vers $\{b\}^*$ associant les mots de même longueur n'est ni reconnaissable ni préfixe-reconnaissable.
La relation $\{(a^nb^k,a^nc^l) \mid k,l,n \geq 0\}$ est la clôture à gauche par a^* de la relation reconnaissable $b^* \times c^*$.

1.4.2 Relations rationnelles (et quelques restrictions)

Nous avons vu au début de ce paragraphe la définition de la loi de composition interne du monoïde produit. Les opérations d'union et d'étoile sur les parties de $\Sigma^* \times \Sigma^*$ sont définies de la façon habituelle. Grâce à ces opérations, on peut donc former des expressions rationnelles sur ce monoïde représentant des ensembles

rationnels de paires de mots, que nous appellerons relations ou transductions rationnelles.

En tant que parties rationnelles d'un monoïde, les relations rationnelles peuvent aussi être caractérisées à l'aide d'automates sur $\Sigma^* \times \Sigma^*$ appelés transducteurs. On se restreint en général pour plus de simplicité à des transducteurs dont les étiquettes sont dans l'ensemble générateur $(\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\})$. Une transition étiquetée par (a,b) entre les états de contrôle p et q d'un transducteur est notée $p \xrightarrow{a/b} q$.

Exemple 1.25. La relation $\{(a^n,v) \mid n \geq 0, v \in b^*(ab^*)^n\}$ est une transduction rationnelle, qui peut être représentée par l'expression rationnelle

$$((\varepsilon,b)^*(a,a))^*(\varepsilon,b)^* = \{(a,a),(\varepsilon,b)\}^*.$$

Elle est aussi la relation acceptée par un transducteur à un état possédant deux boucles étiquetées respectivement par a/a et ε/b.

Nous ne distinguerons pas un transducteur de la relation qu'il accepte, et écrirons simplement $(w,w') \in T$ quand la paire (w,w') est acceptée par un certain transducteur T. Le *domaine* $\mathrm{Dom}(T)$ et le *co-domaine* (ou image) $\mathrm{Im}(T)$ de T sont les ensembles $\{w \mid (w,w') \in T\}$ et $\{w' \mid (w,w') \in T\}$. Un transducteur reconnaissant une relation qui est une fonction est qualifié de *fonctionnel*. Un transducteur est *non-ambigu* s'il possède au plus un calcul acceptant sur toute paire (u,v).

1.4.2.1 Relations lettre-à-lettre et relations synchronisées

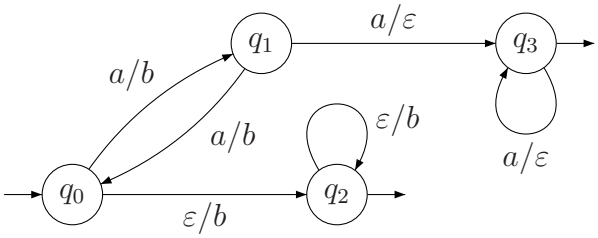
Des classes intéressantes de relations rationnelles peuvent être obtenues en forçant une forme de synchronisation entre la façon dont un transducteur consomme son entrée et produit sa sortie. En particulier, les relations rationnelles sur les paires de mots de la même longueur, appelées relations lettre-à-lettre ou alphabétiques, coïncident avec les parties rationnelles du monoïde libre $(\Sigma \times \Sigma)^*$, et sont donc reconnues par les automates finis à étiquettes dans $\Sigma \times \Sigma$ (plus éventuellement $(\varepsilon,\varepsilon)$) appelés transducteurs lettre-à-lettre ou synchrones.

Exemple 1.26. La relation $\{(a^n,b^n) \mid n \geq 0\}$ est une transduction lettre-à-lettre. Elle est acceptée par le transducteur à un état possédant une unique boucle étiquetée par a/b.

Une forme plus lâche de synchronisation introduite dans [EM65] autorise l'usage d'un symbole spécial $\Box$ à ajouter (uniquement) à la fin du mot le plus court d'une paire de mots afin que la paire en question puisse être accepté par un transducteur synchrone. Ces relations sont en général appelées *automatiques* ou définissables par automates. Plus formellement, une relation R est automatique si la relation $\{(u\Box^i, v\Box^j) \mid (u,v) \in R, i = \max(0,|v| - |u|), j = \max(0,|u| - |v|)\}$ est une transduction lettre-à-lettre.

Cette classe de relations a également été étudiée dans [FS93] sous le nom de *relations synchronisées*. Une relation est synchronisée à gauche si elle peut s'écrire comme une union finie de produits $R.S$, où R est une transduction lettre-à-lettre et S est de la forme (L,ε) ou (ε,L), avec L un langage régulier sur Σ. De façon équivalente, les relations synchronisées sont celles qui sont acceptées par les transducteurs dans lesquels pour tout chemin $q_0 \xrightarrow{x_0/y_0} q_1 \dots q_{n-1} \xrightarrow{x_n/y_n} q_n$, il existe $k \in [0,n]$ tel que pour tout $i \in [0,k-1]$, $x_i,y_i \in \Sigma$ et soit $x_k = \dots = x_n = \varepsilon$ soit $y_k = \dots = y_n = \varepsilon$. De tels transducteurs sont dits synchronisés à gauche. Les relations et les transducteurs synchronisés à droite sont définis de façon similaire. Bien sûr, les transductions synchronisées incluent strictement les clôtures rationnelles de relations reconnaissables, mais sont strictement incluses dans les transductions rationnelles générales (par exemple, la relation de l'exemple 1.25 n'est acceptée par aucun transducteur synchronisé).

Exemple 1.27. La relation $R = \{(a^{2k},b^{m>2k}),(a^{2k+1},b^{m\leq 2k}) \mid k \geq 0\}$ est synchronisée à gauche. Elle est acceptée par le transducteur suivant :



Remarque 1.28. Si on applique la procédure de déterminisation standard à un transducteur synchrone, on obtient un transducteur synchrone non-ambigu équivalent (c'est à dire possédant au plus un chemin acceptant par paire de mots). Ce résultat reste vrai pour les transducteurs synchronisés.

1.4.2.2 Relations séquentielles

Si l'on considère les transducteurs comme des automates lisant un mot en entrée (la première composante de la paire de mots lue) et produisant un mot en sortie (la seconde composante), la notion standard de déterminisme des automates n'a pas beaucoup d'intérêt dans ce cas, parce qu'elle s'appuie à la fois sur l'entrée et sur la sortie. Par exemple, un transducteur possédant les transitions $q_0 \xrightarrow{a/b} q_1$ et $q_0 \xrightarrow{a/c} q_2$ pourrait être déterministe, ce qui ne correspond pas à l'intuition. Une notion plus adaptée est celle de séquentialité, qui fait uniquement référence à la façon dont un transducteur consomme son entrée.

Un transducteur T d'ensemble d'états Q est *séquentiel* si pour tous $q,q' \in Q$, si $q \xrightarrow{x/y} q'$ et $q \xrightarrow{x'/y'} q''$ alors soit $x = x'$, $y = y'$ et $q' = q''$, soit $x \neq \varepsilon$, $x' \neq \varepsilon$ et $x \neq x'$. En d'autres termes, un transducteur est séquentiel si l'automate obtenu

en effaçant la seconde composante de chacune de ses transitions est déterministe. Remarquez qu'un transducteur séquentiel est nécessairement fonctionnel.

1.4.2.3 Propriétés des relations et transducteurs rationnels

Il existe un rapport étroit entre les langages réguliers (rationnels) et les relations rationnelles. En particulier, en effaçant la première (resp. la seconde) composante de chaque transition d'un transducteur on obtient un automate fini reconnaissant le domaine (resp. l'image) de la relation associée. Donc le domaine et l'image d'une relation rationnelle sont des langages réguliers. D'autre part, une construction simple permet de restreindre le domaine ou l'image d'une transduction à un ensemble régulier, et cette construction préserve la séquentialité ou la synchronie du transducteur.

Lemme 1.29. *Les relations rationnelles ont un domaine et une image réguliers, et sont fermées par restriction à un domaine régulier ou une image régulière. De plus, si R est un langage régulier sur Σ, pour tout transducteur séquentiel (resp. synchrone) T sur Σ, le transducteur obtenu en restreignant T au domaine ou à l'image R est séquentiel (resp. synchrone).*

Une conséquence très intéressante de ces résultats est également que l'image d'un langage régulier par une transduction rationnelle est aussi régulière : en effet, partant d'un tel langage L, si l'on restreint le domaine d'un transducteur T à L, il est facile de construire l'automate fini acceptant l'image de ce nouveau transducteur. Le langage de cet automate est précisément $T(L)$. On dit que les relations rationnelles *préservent* la régularité. Comme les relations rationnelles sont closes par inverse, l'image inverse d'un langage régulier par une transduction rationnelle est elle aussi bien sûr régulière.

Les transductions rationnelles sont closes par composition et par union, mais ni par complément, ni par intersection. À l'opposée, les relations synchronisées forment une algèbre de Boole. Elles sont en fait la plus grande famille connue de relations rationnelles possédant cette propriété. Notons aussi qu'aucune famille de relations au moins aussi générales que les transductions lettre-à-lettre n'est fermée par composition itérée, puisque ces relations peuvent très facilement représenter des transitions de machines de Turing déterministes.

1.4.3 Extensions aux termes

La plupart des familles de relations binaires sur les mots ont été étendues avec un succès certain au cas des termes. Cependant, de la même façon que pour les langages de termes, la situation est bien moins claire d'un point de vue algébrique. En particulier, il n'existe pas réellement de notion de rationalité pour les relations de termes.

1.4.3.1 Relations reconnaissables

Par analogie avec les relations reconnaissables de mots, on peut définir une relation reconnaissable de termes comme l'ensemble de toutes les couples de termes acceptées par une union finie de couples d'automates d'arbres finis. Par exemple, pour $i \in [1,n]$, soit (A_i,B_i) un couple d'automates d'arbres descendants non-déterministes. La relation $R = \{(s,t) \mid \exists i \in [1,n], s \in L(A_i) \wedge t \in L(B_i)\}$ est par définition reconnaissable. Les relations reconnaissables ont le même manque d'expressivité sur les termes que sur les mots, et ne mettent pas en rapport les termes eux-mêmes mais plutôt des langages entiers.

1.4.3.2 Transductions closes

Une extension intéressante des relations reconnaissables généralise les clôtures à gauche ou à droite de relations reconnaissables de mots (cf. § 1.4.1) au cas des termes. De telles relations sont appelées *transductions closes d'arbres* (ou *ground tree transductions*) et ont été définies dans [DTHL87], accompagnées d'une classe d'automates les caractérisant appelés transducteurs clos ou *ground tree transducers* (GTT). Une relation R est une transduction close si elle est de la forme

$$R = \{(c[s_1 \dots s_n], c[t_1 \dots t_n]) \mid c \in C_n(F) \wedge \forall i \in [1,n], (s_i,t_i) \in R'\}$$

où R' est une relation reconnaissable. Les transductions closes furent initialement définies pour aider à l'étude des systèmes de réécriture clos de termes. Elles sont fermées par inverse, par composition et par clôture transitive de la composition, mais pas par union ni par complémentaire. Ceci rappelle les clôtures de relations reconnaissables de mots. Une question intéressante serait d'étendre les transductions closes dans le même esprit que les clôtures rationnelles des relations reconnaissables de mots, dans l'espoir d'obtenir certaines de leurs bonnes propriétés de clôture.

1.4.3.3 Relations synchronisées

Une autre classe de relations qui peut être vue comme une généralisation du cas des mots est celle des relations automatiques (cf. § 1.4.2). De la même façon que les relations automatiques de mots utilisent un symbole spécial $\Box$ pour « égaliser » deux mots de taille différente avant de les faire accepter par un automate, nous utiliserons le symbole $\perp \notin F$ pour définir la *superposition* de deux termes t et s dans $T(F)$, notée $[ts]$. Tout d'abord, soit $F_\times$ l'alphabet gradué $\{fg \mid f,g \in (F \cup \{\perp\})\}$, où l'arité de fg est le maximum entre celle de f et celle de g. Pour $t = f(t_1 \dots t_n)$ et $s = g(s_1 \dots s_m)$, on définit $[ts]$ récursivement comme

$$[f(t_1 \dots t_n), g(s_1 \dots s_m)] = fg([t_1 s_1] \dots [t_k s_k])$$

où $k = \max(n,m)$, $\forall i \in [n+1,k], t_i = \perp$ et $\forall i \in [m+1,k], s_i = \perp$. Cette opération est étendue aux ensembles de termes de la façon habituelle. Nous appelons *relation automatique* (ou *synchronisée*) de termes toute relation binaire de la forme $R = \{(s,t) \mid [st] \in L\}$, où L est un langage régulier de termes sur $F_\times$. Restreintes aux mots, ces relations coïncident avec les relations automatiques comme définies précédemment (c'est à dire les relations acceptées par les transducteurs synchronisés). Elles ont la même propriété d'être closes par les opérations booléennes.

1.4.3.4 Relations rationnelles ?

Dans notre comparaison entre les relations binaires de mots et d'arbres, la dernière famille non évoquée est celle des relations rationnelles. Malheureusement, la notion de rationalité est moins claire dans le cas des relations de termes que dans celui des relations de mots. Si les relations rationnelles de mots bénéficient d'une définition algébrique claire comme les parties rationnelles d'un monoïde, une telle notion n'a à ce jour pas été adoptée pour les termes. Pour paraphraser une revue des relations de termes proposée par Raoult [Rao91], plusieurs critères doivent être pris en compte si l'on souhaite évaluer la pertinence d'une famille de relations en tant qu'extension la plus naturelle des mots aux termes :

1. Restreinte aux mots, la famille coïncide-t-elle avec les relation rationnelles ? Contient-elle au moins la relation identité ?
2. La famille est-elle close par composition ? Par inverse ?
3. La famille est-elle « robuste » (admet-elle plusieurs caractérisations naturelles équivalentes) ?
4. Les relations de la famille préservent-elles les ensembles réguliers (c'est à dire, associent à tout ensemble régulier de termes un autre ensemble régulier) ? Ont-elles un domaine et une image réguliers ?

Les relations rationnelles vérifient chacune de ces propriétés. Certaines des questions posées dans [Rao91] ne sont pas reprises ici, mais l'idée qu'il en ressort est que jusqu'à présent, aucune des relations de termes existantes n'a satisfait tous ces critères de manière satisfaisante.

En dépit de ce constat, plusieurs familles de relations cherchant à généraliser les relations rationnelles de mots ont vu le jour et ont chacune apporté d'intéressants résultats et applications. Nous dirigeons le lecteur intéressé vers l'article précédemment cité ainsi que [GS97, CDG$^+$] pour de plus amples détails. Dans le chapitre 3, nous utiliserons une notion de relations définie dans [Rao97] qui est plus générale que les familles les plus courantes de relations sur les termes, mais conserve cependant d'intéressantes propriétés et est exprimée à l'aide de grammaire de graphes.

1.5 Logiques sur les graphes

On désigne sous le terme de logiques des langages mathématiques utilisés pour exprimer certaines propriétés d'objets comme les mots, les termes, les graphes, et de façon plus générale tous types de *structures relationnelles*. Nous présentons deux principaux types de logiques sur les graphes coloriés, la bien connue logique du premier ordre et la restriction monadique de la logique du second ordre, puis présentons certaines de leurs variantes. Ce paragraphe est bien entendu très superficiel, et n'a pour seul but que de fixer les notations et la terminologie.

1.5.1 Logique du premier ordre

Soit $X = \{x,y,\ldots\}$ un ensemble dénombrable de variables destinées à être instanciées par des sommets de graphes. Les formules de logique du premier ordre sur les graphes à étiquettes dans Σ et à couleurs dans C sont soit :

- une formule atomique de la forme *true*, $x = y$, $x \xrightarrow{a} y$ ou $c(x)$ avec $a \in \Sigma$, $c \in C$ et $x,y \in X$,
- une conjonction $\phi \wedge \psi$ de deux formules ϕ et ψ ou la négation $\neg\phi$ d'une formule ϕ,
- une quantification existentielle $\exists x\phi$.

Étant donné un graphe colorié (G,K), une *valuation* est une application partielle de l'ensemble X des variables vers l'ensemble V_G des sommets de G associant un et un seul sommet à certaines des variables de X. Les formules atomiques $x = y$, $x \xrightarrow{a} y$ et $c(x)$ sont satisfaites dans (G,K) sous la valuation γ si l'on a respectivement $\gamma(x) = \gamma(y)$, $\gamma(x) \xrightarrow[G]{a} \gamma(y)$ et $K(\gamma(x)) = c$. Une conjonction $\phi\wedge\psi$ est satisfaite si à la fois ϕ et ψ le sont, et une négation $\neg\phi$ n'est satisfaite que si ϕ ne l'est pas. Enfin, une formule existentiellement quantifiée $\exists x\phi$ est satisfaite s'il existe un sommet $v \in V_G$ tel que ϕ soit satisfaite sous la valuation γ', où $\gamma'(y) = \gamma(y)$ pour tout $y \neq x$ et $\gamma'(x) = v$. La formule spéciale *true* est satisfaite par tous les graphes sous toute valuation. La satisfaction d'une formule ϕ dans (G,K) sous la valuation γ est notée $(G,K),\gamma \models \phi$.

D'autres connecteurs utiles peuvent être définis à partir de cette syntaxe. On définit la disjonction $\phi \vee \psi$ comme un raccourci d'écriture pour la formule $\neg(\neg\phi \wedge \neg\psi)$, et la quantification universelle $\forall x\phi$ comme $\neg\exists\neg\phi$. Dans une formule de la forme $\exists x\phi$ ou $\forall x\phi$, toutes les occurrences de la variable x dans ϕ sont qualifiées de *liées*. Les variables qui ont des occurrences non liées dans une formule sont appelées variables *libres*. On écrit parfois $\phi(x_1,\ldots x_n)$ pour insister sur le fait qu'au plus les variables x_1 à x_n sont libres dans ϕ. Une formule dans laquelle aucune variable n'est libre est dite *close*. La satisfaction dans (G,K) d'une formule close ϕ ne dépend pas d'une quelconque valuation des variables, elle est donc simplement notée $(G,K) \models \phi$, et on dit que (G,K) est un *modèle* de ϕ.

L'ensemble $L(\phi)$ des modèles d'une formule close ϕ donnée est parfois appelé ensemble *défini* par ϕ. Dans le cas de graphes représentant des mots ou des termes, on parle plutôt du *langage* de ϕ. De tels ensembles sont qualifiés de définissables au premier ordre. Réciproquement, l'ensemble de toutes les formules closes du premier ordre satisfaites par un graphe (G,K) donné est appelé *théorie au premier ordre* de (G,K), notée $Th(G,K)$. On dit que la théorie de (G,K) est décidable si cet ensemble est récursif, c'est à dire s'il existe un algorithme effectif permettant de décider si une formule donnée est satisfaite sur (G,K) ou non.

La logique du premier ordre exprime principalement des propriétés « locales » des graphes. En particulier, il lui est impossible d'exprimer par exemple l'existence d'un chemin reliant deux sommets donnés d'un graphe. Néanmoins, il a été montré [Tra50] que le problème de savoir si une formule du premier ordre quelconque possède au moins un modèle dans l'ensemble des graphes finis est indécidable.

1.5.2 Logique du second ordre monadique

Nous présentons maintenant une logique très expressive sur les graphes appelée logique du second ordre monadique (MSO). La logique du second ordre est construite de façon similaire à la logique du premier ordre, à ceci près que la quantification sur des variables représentant des relations d'arité quelconque entre sommets est autorisée. Dans la logique MSO, le mot monadique se rapporte au fait qu'on ne considère que des variables représentant des relations d'arité 1, c'est à dire des ensembles de sommets.

Formellement, la logique du second ordre monadique étend la syntaxe de la logique du premier ordre avec un ensemble dénombrable de variables d'ordre supérieur $X,Y,X_1,\ldots$ représentant des ensembles de sommets de graphes, ainsi qu'avec de nouveaux prédicats atomiques de la forme $x \in X$, où x est une variable du premier ordre et X une variable d'ensemble. La quantification sur les variables d'ensembles est elle aussi permise, et les valuations incluent les deux types de variables.

La logique du second ordre monadique permet par exemple la description de propriétés de chemins dans les graphes, comme l'existence de chemins avec certaines étiquettes entre deux sommets, la confluence de chemins, etc.

1.5.3 Entre premier ordre et second ordre monadique

Une extension courante de la logique du premier ordre sur les graphes Σ-étiquetés est d'ajouter à sa syntaxe différents types de prédicats d'accessibilité. La plus faible de telles extensions utilise des prédicats de la forme $x \xrightarrow{*} y$, qui expriment l'existence d'un chemin dirigé d'étiquette quelconque entre les sommets désignés par x et y sous l'interprétation courante des variables. Des extensions

plus expressives incluent des prédicats de la forme $x \xrightarrow{A^*} y$, où A est un sous-ensemble de Σ, ou même $x \xrightarrow{L} y$ où L est un langage régulier dans Σ^*. Ces deux derniers types de prédicats ajoutent la possibilité d'énoncer l'existence d'un chemin avec une certaine étiquette entre x et y, soit en restreignant l'ensemble d'étiquettes autorisées, soit en se restreignant directement à un ensemble régulier d'étiquettes de chemin.

Un extension plus générale encore de la logique du premier ordre consiste à ajouter à la logique un opérateur unaire de fermeture transitive, TC. Étant donné une formule du premier ordre $\phi(x,y)$ ayant deux variables libres x et y, on définit récursivement $TC(\phi)$ comme $\exists z \phi(x,z) \wedge TC(\phi)(z,y)$. Cette nouvelle logique est strictement plus expressive que les extensions précédemment citées, tout en restant exprimable en logique du second ordre monadique.

1.5.4 Logiques temporelles

Les logiques temporelles sont des logiques modales spécialisées dans la description de propriétés qui se rapportent au passage du temps dans un système. Lorsque l'on s'intéresse à ce genre de logiques sur les graphes, le temps est supposé augmenter d'une unité à chaque fois qu'une transition est suivie. Par conséquent, les logiques temporelles expriment plutôt des propriétés de l'arbre de dépliage d'un graphe (cf. § 2.1.2.1) plutôt qu'au graphe lui-même. Toutes les logiques temporelles mentionnées ici sont facilement traduisibles en logique du second ordre monadique. En fait, toutes les principales logiques temporelles sont subsumées par une logique modale très expressive appelée μ-calcul, une logique de point fixe d'expressivité équivalente au fragment invariant par bisimulation de la logique du second ordre monadique (voir [JW96]), qui ne sera pas présentée ici.

1.5.4.1 CTL*

La logique CTL* (où CTL signifie *Computation Tree Logic*, ou logique des arbres d'exécution) est une logique capable d'exprimer des propriétés de plusieurs exécutions possibles au sein d'un même système (cf. [Eme90]). Ses formules sont de deux types, des formules d'états et des formules de chemins. Les formules d'états sont construites selon la syntaxe suivante :

$$\phi = A\psi \mid E\psi \mid c \in C \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

où C est l'ensemble de couleurs de l'arbre considéré, ψ est une formule de chemin et ϕ,ϕ_1,ϕ_2 des formules d'état. Les formules de chemins sont construites de la façon suivante :

$$\psi = \phi \mid X\psi \mid \psi_1 U \psi_2 \mid \neg\psi \mid \psi_1 \wedge \psi_2$$

où ϕ est une formule d'état (toute formule d'état est aussi une formule de chemin) et ψ,ψ_1,ψ_2 sont aussi des formules de chemin.

Les formules CTL* bien formées sont des formules d'état, qui sont évaluées à la racine d'un arbre colorié t. Les propositions atomiques (couleurs) sont satisfaites si la racine de l'arbre courant (le « temps présent » dans le point de vue temporel) est étiquetée par la couleur en question. Les quantificateurs A et E portent sur l'ensemble des branches de t, c'est à dire les chemins de longueur maximale dans l'arbre dont l'origine est la racine. Les formules d'état $\mathsf{A}\phi$ et $\mathsf{E}\phi$ expriment le fait que la formule de chemin ϕ est satisfaite dans t par rapport à tous les chemins (resp. au moins un des chemins) de t d'origine la racine de t.

Les formules de chemin sont évaluées par rapport à un chemin π donné vu comme la séquence de sommets coloriés. On note π^i le chemin π privé de ses i premiers sommets. Une formule de chemin ψ consistant simplement en une formule d'état est satisfaite dans t par rapport à π si elle est vraie dans le sous-arbre de t dont la racine est le premier sommet de π. La formule $\mathsf{X}\psi$ (lire *next* ψ, ou *à l'instant suivant* ψ) est satisfaite si ψ est satisfaite par rapport à π^1. La formule $\psi_1\mathsf{U}\psi_2$ (lire ψ_1 *until* ψ_2, ou ψ_1 *jusqu'à ce que* ψ_2) est satisfaite s'il existe un entier $k \geq 0$ tel que ψ_2 est satisfaite par rapport à π^k, et ψ_1 est satisfaite par rapport à chacun des π^j pour $j < k$ (en d'autres termes, ψ_2 sera vraie à un certain moment dans le futur, et ψ_1 doit être vraie à chaque instant jusque là).

Les opérateurs temporels classiques F (*finally*, ou *un jour*) et G (*globally* ou *toujours*) peuvent être exprimés à l'aide de l'opérateur U. Les formules $\mathsf{F}\phi$ et $\mathsf{G}\phi$ sont respectivement équivalentes à $\mathit{true}\mathsf{U}\phi$ et $\neg\mathit{true}\mathsf{U}\neg\phi$.

1.5.4.2 LTL et CTL

Les logiques LTL (*linear time logic*, logique du temps linéaire) [Pnu77] et CTL [CES86] sont des fragments intéressants (et antérieurs) de CTL*. Les formules LTL sont toutes les formules CTL* sans quantificateur, qui sont implicitement évaluées sur tous les chemins partant de la racine d'un arbre colorié (un quantificateur A implicite est ajouté). Elles peuvent uniquement décrire la structure générale de l'ensemble des branches d'un arbre. Les formules CTL quant à elles peuvent être définies comme les formules CTL* dans lesquelles il existe une alternance stricte entre les quantificateurs de chemins et les opérateurs temporels. Ces deux fragments sont strictement moins expressifs que la logique CTL*, mais sont incomparables entre eux.

1.6 Notions de *model checking* fini

Le problème général du *model checking*, ou vérification de modèles, est étant donné un système S et une formule close ϕ dans une logique appropriée, de dé-

terminer si S est un modèle pour ϕ, ou en d'autres termes si le système S vérifie la propriété énoncée par ϕ. Par exemple on peut imaginer vouloir vérifier, étant donné un automate fini modélisant une abstraction du comportement d'un programme et une formule LTL exprimant une propriété quelconque, si l'arbre de dépliage de l'automate satisfait la formule, et si possible en déduire si le programme que l'on a modélisé possède la propriété correspondante.

La dénomination générale de vérification de modèles ne fait pas référence à une catégorie précise d'algorithmes ou de techniques, mais plutôt au problème théorique de satisfaction logique lui-même. Les détails des techniques existantes pour résoudre ce problème varient considérablement selon la logique et la famille de modèles que l'on considère. Cependant, dans le cas des systèmes à nombre d'états fini (ou leurs arbres de dépliage) et la vérification de leurs propriétés temporelles, des méthodes assez répandues et efficaces existent, communément réunies sous l'appellation « vérification par la théorie des automates ».

Dans plusieurs cas, et en particulier pour les formules CTL*, CTL et LTL, l'ensemble des modèles (arbres) d'une formule donnée peut être représenté à l'aide de certaines classes d'automates finis sur les arbres infinis (ou même les mots infinis dans le cas de LTL). Dans ce cas, le problème de vérification de modèles consiste juste en un test d'appartenance d'un élément au langage d'un de ces automates. D'autres problèmes comme ceux de la validité ou satisfaisabilité d'une formule se traduisent quant à eux en tests d'universalité ou test du vide d'un langage. On peut aussi s'intéresser à des questions plus générales du type « Les modèles de la formule ϕ sont-ils tous des modèles de ψ ? » ou « Dans l'ensemble S, certains éléments sont-ils des modèles de ϕ, et si oui lesquels ? ». De telles questions se traduisent directement en problèmes d'inclusion ou d'intersection de langages.

L'approche de la vérification par la théorie des automates est un sujet très riche, qui dépasse pour la plus grande part de la portée de cette thèse (pour plus de détails, consulter par exemple [VW86, CES86]). La principale idée que nous souhaitons mettre en avant est que des propriétés de programmes exprimées comme des formules de logique temporelle peuvent la plupart du temps être traduites en automates dont les langages sont les ensembles de modèles de ces formules.

Chapitre 2

Graphes infinis

Comme nous l'avons mentionné dans le chapitre précédent, les graphes étiquetés sont caractérisés par des ensembles finis de relations binaires sur un domaine dénombrable, où chaque sommet représente un élément du domaine et les relations entre ces éléments sont figurées par les arcs. Une conséquence de cette définition très générale est qu'une très grande variété de formalismes exploitant des relations binaires peut s'exprimer et se représenter en termes de graphes.

Dans ce chapitre, nous commençons par présenter plusieurs façons possibles de définir des familles de graphes infinis de présentation finie, soit en en définissant un ensemble de représentants, auquel cas on parle de représentation interne, soit en décrivant directement la structure de ces graphes sans nommer leurs sommets (on parle alors de représentation externe). Dans un second temps nous énumérons certaines des familles de graphes infinis les plus connues, explicitons leurs différentes représentations internes et externes, et citons certaines de leurs principales propriétés. Enfin, nous donnons quelques indices sur des similitudes entre l'étude des graphes infinis et quelques pistes récentes dans le domaine de la vérification automatique de systèmes infinis.

2.1 Présentations finies de graphes infinis

Nous énumérons ici certains des moyens par lesquels des familles de graphes infinis peuvent être décrites au moyen de formalismes finis. On peut établir une distinction entre deux grands types d'approche. La première façon de décrire un graphe, que nous qualifierons de représentation interne, consiste à spécifier directement un ensemble fini de relations binaires, une pour chaque lettre de l'alphabet d'étiquettes, entre les éléments d'un certain domaine, par exemple des mots ou des termes sur un certain alphabet.

Une autre catégorie de présentations finies, qualifiées d'externes, consiste à décrire directement la structure des graphes, soit en les caractérisant comme

le résultat de transformations structurelles à partir d'une famille finie (ou déjà connue) de graphes, soit comme l'ensemble des solutions d'un système d'équations dont les inconnues sont des graphes, soit comme l'ensemble de modèles d'une formule close dans un langage logique approprié, etc.

Nous donnons ici une brève description des plus importantes de ces caractérisations, en insistant tout particulièrement sur les présentations internes qui se rapportent le plus au présent travail.

2.1.1 Présentations internes

Comme précédemment rappelé, un graphe étiqueté sur Σ peut être défini simplement comme un système de transition étiqueté, ou en d'autres termes comme un ensemble fini de relations binaires sur le même domaine, chaque relation étant associée à un symbole dans Σ. Donc réciproquement, tout ensemble de telles relations est aussi implicitement associé à un graphe. Nous ne faisons pas de distinction entre ces deux notions. Dans ce cadre, étudier des familles de graphes (ou plus précisément des familles de représentants de graphes) revient à étudier des familles de relations binaires.

Comme nous ne nous intéressons qu'à des relations de présentation finie, ainsi qu'aux liens entre les graphes et d'autres domaines de l'informatique (comme la théorie des langages ou de la réécriture et la vérification de programmes), la manière la plus simple de caractériser de façon finie des familles de graphes infinis est de réutiliser des formalismes existant servant à définir des relations binaires, et en particulier les machines (et automates) et les systèmes de réécriture.

2.1.1.1 Graphes définis par des machines ou des automates

Dans ce paragraphe, étant donné un alphabet fini Σ, nous désignons par M un accepteur (ou machine) quelconque pour des ensembles de mots dans Σ^* induisant une relation de transition $(\xrightarrow[M]{a})_{a\in(\Sigma\cup\{\varepsilon\})}$ étiquetée par $(\Sigma\cup\{\varepsilon\})$ sur son ensemble de configurations. M pourrait être une machine de Turing, un automate à pile, ou virtuellement tout type de formalisme. Par simplicité, nous appellerons M une *machine*, sans autre précision. Nous donnons une brève description de plusieurs méthodes standard permettant d'associer un graphe infini à une telle machine.

Graphes des configurations. La relation de transition de la machine M peut être vue directement comme un système de transition étiqueté sur $\Sigma\cup\{\varepsilon\}$. Nous pouvons donc également voir cet ensemble de relations comme un graphe étiqueté sur $(\Sigma\cup\{\varepsilon\})$, que nous appelons *graphe des configurations* de M et notons K_M.

Les sommets de K_M sont les configurations possibles de M et ses arcs représentent les transitions de M d'une configuration à l'autre, ε-transitions incluses.

Ceci constitue la plus simple façon d'associer un graphe à une machine, est utilisée en particulier pour associer un graphe fini à la description formelle d'un automate. En fait, la nuance entre les graphes et les automates (ou machines) ne tient qu'à la spécification d'ensembles de configurations initiales et finales.

Graphes de transition. Le graphe de configuration d'une machine M fait apparaître chaque transition comme un arc. Pour la plupart des familles de machines possédant un ensemble infini de configurations, le graphe des configurations peut donc contenir de nombreux arcs étiquetés par ε. Lorsqu'on considère ces transitions comme des étapes de calcul internes, il peut être intéressant de chercher à dissimuler ces transitions et de ne faire apparaître que le comportement observable de M depuis l'extérieur, c'est à dire la façon dont M lit des symboles en entrée. Ceci peut être fait par transformation du graphe K_M. Cependant, cette transformation doit être effectuée en restant attentif à ne pas introduire de nouvelles étiquettes de chemins ni en détruisant la structure globale de K_M.

Dans le cas des automates finis, nous avons mentionné que tout automate contenant des ε-transitions peut être transformé en un automate équivalent, c'est à dire acceptant le même langage, qui ne contienne aucune ε-transition. Nous généralisons cette idée au graphe des configurations d'une machine M quelconque, et appelons le résultat obtenu *graphe de transition* de M.

Avant de donner la définition du graphe de transition, nous établissons une distinction entre configurations *internes* et configurations *externes* d'une machine M, de la même façon que nous distinguons les transitions internes (ε-transitions) des transitions externes. Une configuration c de M est qualifiée d'externe si elle est la source d'au moins une transition étiquetée par un certain $a \in \Sigma$, ou si c'est une configuration puits (c'est à dire si le sommet correspondant de K_M est de degré sortant 0). Elle est interne dans tous les autres cas. Nous notons C_e l'ensemble des configurations externes de M.

Définition 2.1. Soit M un accepteur de mots sur Σ avec ε-transitions, le *graphe de transition* de M est

$$G_M \;=\; \big\{(c_1,a,c_2) \mid c_1 \xrightarrow[M]{a\varepsilon^*} c_2,\; a \in \Sigma\; c_1,c_2 \in C_e\big\}.$$

Pour que cette définition ait un sens, nous devons poser quelques hypothèses supplémentaires sur les accepteurs de mots que nous considérons. En particulier, puisque le but du graphe de transition est d'illustrer de façon plus synthétique la structure des calculs visibles de machines, il est important qu'un graphe de transition soit *fidèle* à la machine correspondante et à son graphe des configurations, au sens où il devrait respecter la relation d'accessibilité entre configurations, et où chaque configuration initiale ou finale devrait apparaître dans le graphe.

Définition 2.2. Le graphe de transition G_M d'une machine M sur Σ est *fidèle* si pour tout couple (c_1,c_2) de configurations externes de M et pour tout $u \in$

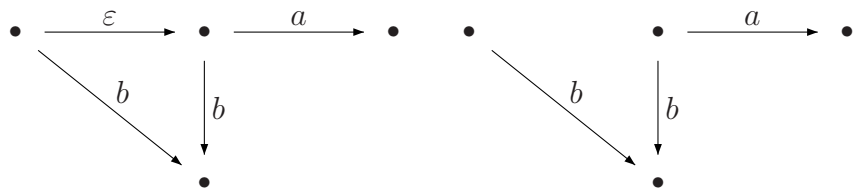


FIG. 2.1 – *Graphe des configurations et graphe de transition (non fidèle) associé.*

Σ^*, $c_1 \xrightarrow[C_M]{u} c_2 \implies c_1 \xrightarrow[G_M]{u} c_2$, et si la configuration initiale c_0 et toutes les configurations acceptantes de M accessibles depuis c_0 sont des sommets de G_M.

La figure 2.1 montre le graphe des configurations d'une machine et le graphe de transition non fidèle associé. Une condition suffisante pour que le graphe de transition d'une machine M soit fidèle est que toute configuration externe de M ne soit source d'aucune ε-transition, et que ses configurations initiale et finales soient toutes externes. On parle dans ce cas d'une machine *normalisée*. Intuitivement, cela signifie qu'à tout point d'un calcul, la machine peut être soit dans un mode stable, c'est à dire en attente de la lecture d'un symbole ou en fin de calcul (soit acceptant, soit non-acceptant), soit dans un mode instable, effectuant des calculs internes initiées par les lectures précédentes. Cette notion a été proposée par Stirling dans [Sti00] pour permettre la représentation des graphes préfixe-reconnaissables comme les graphes de transition des automates à pile normalisés. Elle fut plus tard réutilisée dans [Car01] pour généraliser ce résultat aux graphes de transition des automates à pile d'ordre supérieur.

Proposition 2.3. *Les graphes de transition des machines normalisées sont fidèles.*

Démonstration. Considérons une machine normalisée M sur Σ, son graphe des configurations K_M et son graphe de transition G_M. Soit (c_1,c_2) un couple de configurations externes quelconque de M tel que $c_1 \xrightarrow[C_M]{u} c_2$ pour une certaine étiquette de chemin $u \in \Sigma^*$. Considérons un chemin quelconque ρ étiqueté par u entre c_1 et c_2 dans K_M, et montrons par récurrence sur la longueur de ρ que $c_1 \xrightarrow[G_M]{u} c_2$.

Le cas où ρ est de longueur 0 est trivial puisque, comme $u = \varepsilon$, $c_1 = c_2$. Supposons que ρ soit de longueur $n \geq 1$. Si ρ commence par une ε-transition, ceci contredit le fait que c_1 est une configuration externe et que M est normalisée. Donc ρ doit commencer par une transition externe étiquetée par un certain $a \in \Sigma$. Soit c_3 la première configuration externe distincte de c_1 traversée par ρ. Par définition des configurations externes, la portion du chemin ρ entre c_1 et c_3 doit consister en un arc étiqueté par a suivi d'un nombre quelconque d'arcs étiqueté par ε. Donc par définition des graphes de transition, il doit exister un arc étiqueté par a entre

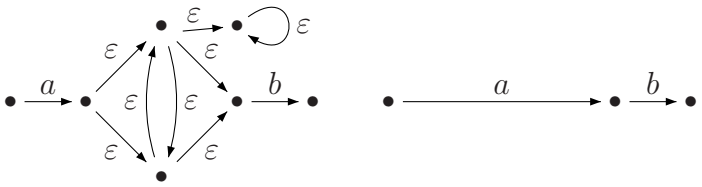


FIG. 2.2 – *Graphe des configurations et graphe de transition déterministe d'une machine non déterministe.*

c_1 et c_3 dans G_M. Par récurrence sur la portion restante de ρ étiquetée par $a^{-1}u$ entre c_3 et c_2, il doit exister un chemin $c_3 \xrightarrow{a^{-1}u} c_2$ dans G_M, et donc aussi un chemin $c_1 \xrightarrow{u} c_2$. □

Une observation intéressante concernant les graphes de transition est qu'ils ne reflètent que partiellement le déterminisme de la machine qui les définit. Premièrement, une machine déterministe (c'est à dire une machine dont le graphe de configuration est déterministe) a nécessairement un graphe de transition déterministe. La réciproque est moins satisfaisante : même certaines machines non-déterministes peuvent avoir un graphe de transition déterministe, comme on peut le voir sur la figure 2.2. Nous allons voir au chapitre 4 (Proposition 4.60) que les machines normalisées *terminantes* (c'est à dire les machines qui ne possèdent aucun chemin infini non acceptant sur aucun mot d'entrée) peuvent être rendues déterministes si leur graphe de transition est déterministe.

Graphes de calcul. Enfin, considérons des machines qui, au lieu d'accepter des langages de mots, acceptent des relations binaires sur un domaine arbitraire V. Des exemples de telles machines incluent les transducteurs rationnels et les machines de Turing acceptant des couples de mots, pour lesquels V est le monoïde libre sur un alphabet donné. On peut aussi citer tous les types de transducteurs d'arbres, auquel cas V est une algèbre libre. Nous appelons de telles machines simplement des *transducteurs*.

Soit $(T_a)_{a\in\Sigma}$ une famille finie de transducteurs, et $L(T_a)$ le sous-ensemble de $V \times V$ accepté par chaque T_a. Nous définissons le graphe de calcul de $(T_a)_{a\in\Sigma}$ comme le graphe dont un couple de sommet (u,v) est lié par un arc a si et seulement si ce couple est accepté par le transducteur T_a.

Définition 2.4. Soit $T = (T_a)_{a\in\Sigma}$ une famille finie de transducteurs sur le domaine V, on définit le *graphe de calcul* (ou *graphe de transduction*) de T par

$$G_T = \{(u,a,v) \mid a \in \Sigma, (u,v) \in L(T_a)\}.$$

Les graphes de calcul ne sont pas définis directement par la relation de transition de machines comme dans le cas précédent. Chacune de leurs relations d'arcs

est plutôt directement caractérisée par l'ensemble des couples acceptés par un transducteur. Cette dernière façon de définir un graphe à l'aide de machines ou automates de représentation finie est peut-être la plus proche de la définition initiale d'un graphe comme ensemble fini de relations binaires. Cependant, elle n'est pas appropriée pour la modélisation du même genre de situations, et son interprétation intuitive est très différente de l'intuition associée aux graphes de transition par exemple.

La notion de *graphe de calcul* fut introduite dans les premières versions de [Cau03], puis utilisée de façon plus systématique dans [CK02a] (où on parle de *graphe de relation*). Ces travaux montrent que pour les automates à pile et les machines de Turing, les graphes de transition et de calcul coïncident à isomorphisme près.

2.1.1.2 Graphes définis par des systèmes de réécriture

Tout système de réécriture de mots ou de termes étiqueté définit naturellement un graphe dont les relations d'arcs illustrent la relation de réécriture du système. Plus précisément, on appelle *graphe de réécriture* d'un système de réécriture R étiqueté par Σ le graphe $\{(u,a,v) \mid u \xrightarrow[R]{a} v\}$. On peut remarquer que, puisque les relations de transition des machines de Turing sont définies vis-à-vis d'un système de réécriture (cf. § 1.3.2), tous les graphes des configurations des familles d'accepteurs décrits au paragraphe 1.3.1 sont les graphes de réécriture des systèmes décrivant leurs relations de transition respectives.

Une autre manière d'associer un graphe à un système de réécriture de mots donné a été proposée dans [CK98]. Elle repose sur la relation de dérivation d'un système plutôt que sur sa relation de réécriture. On définit le *graphe de type Cayley* d'un système de réécriture de mots de la façon suivante.

Définition 2.5. Le graphe de type Cayley à étiquettes dans Σ d'un système de réécriture de mots R sur Γ, avec $\Sigma \subseteq \Gamma$, est le graphe

$$G_R \;=\; \{(u,a,v) \mid a \in \Sigma,\, u,v \in \mathrm{NF}(R),\, ua \xrightarrow[R]{*} v\}.$$

Cette terminologie vient d'une analogie avec la notion de théorie des groupes de *graphe de Cayley* d'un groupe. On peut définir une notion semblable également pour les systèmes de réécriture de *termes* sur un alphabet gradué F donné en considérant Σ comme un sous-ensemble de F_1, l'ensemble des symboles d'arité 1 de F.

Définition 2.6. Le graphe de type Cayley à étiquettes dans Σ d'un système de réécriture de termes R sur F, avec $\Sigma \subseteq F_1$, est le graphe

$$G_R \;=\; \{(s,a,t) \mid a \in \Gamma,\, s,t \in \mathrm{NF}(R),\, a(s) \xrightarrow[R]{*} t\}.$$

Une différence importante entre les systèmes de réécriture et les accepteurs de langages (ou machines) est que ces premiers n'ont pas de notion de lecture d'une entrée, de configurations internes ou externes, de calculs, ou d'acceptation. Par conséquent, il ne semble pas très significatif d'étendre directement la notion de graphe de transition aux systèmes de réécriture. En un sens cependant, les graphes de type Cayley comblent le fossé entre les systèmes de réécriture et les machines en introduisant des notions de début et fin d'une séquence de dérivation, de configurations stables (formes normales), d'entrées (ajout d'un nouveau symbole en tête d'une forme normale) et de terminaison (obtention d'une forme normale).

2.1.1.3 Autres caractérisations internes

Il existe de nombreuses autres manières de donner une caractérisation finie interne de graphes infinis. Il devrait apparaître clairement à présent que tout ensemble fini de relations binaires peut être vue comme un graphe. Ceci est vrai en particulier de toutes les familles de relations présentées au paragraphe 1.4. On admettra donc explicitement l'expression *graphe reconnaissable* étiqueté sur Σ, par exemple, pour se référer à une union finie de graphes de la forme $\{(s,a,t) \mid s \in S,t \in T\}$, noté $S \xrightarrow{a} T$, où S et T sont des ensembles réguliers sur Γ et $a \in \Sigma$. On étend aussi les opérations de clôture à de tels graphes. La *clôture rationnelle à droite* d'un graphe reconnaissable $G = \bigcup_{i\in[1,n]}(S_i \xrightarrow{a_i} T_i)$ est un graphe de la forme $\bigcup_{i\in[1,n]}\{(su,a_i,tu) \mid s \in S_i,t \in T_i,u \in U_i\}$ où chaque U_i est un ensemble régulier. Quand chaque ensemble U_i est égal à Γ^*, on parle simplement de clôture à droite de G.

On a déjà défini au chapitre précédent la restriction d'un graphe à un ensemble de sommets. Dans le cas où les sommets sont des mots (ou des termes), on pourra considérer la restriction des sommets d'un graphe G à un ensemble régulier L. Dans ce cas, on parle de la *restriction rationnelle* de G à L, notée $G|_L$.

Plus généralement, on pourra considérer des familles de graphes engendrées par tous types de relations de présentation finie sur des domaines dénombrables arbitraires, comme des opérations arithmétiques sur les entiers, des relations entre matrices de symboles ou même entre des graphes finis.

2.1.2 Présentations externes

On énumère à présent plusieurs formalismes permettant de définir des familles de graphes infinis de manière finie sans s'appuyer sur les valeurs particulières (ou nommage) de leurs sommets. À la place, ces méthodes décrivent directement la structure des graphes, soit par des transformations de graphes plus simples, soit en considérant les ensembles de solutions de systèmes d'équations utilisant des opérateurs sur les graphes, soit encore en évaluant des ensembles d'expressions

sur de tels opérateurs. Ces caractérisations sont regroupées sous le qualificatif d'*externes*.

2.1.2.1 Transformations de graphes

Un grand nombre de transformations structurelles sur les graphes peuvent être utilisées pour définir des familles de graphes infinis vérifiant certaines propriétés. Une idée fondamentale est, partant d'un graphe simple appelé *générateur* ou d'une famille simple de générateurs dont les propriétés sont connues, d'obtenir de nouveaux graphes par des transformations préservant ces propriétés. Ce paragraphe présente les plus courantes de ces transformations.

Nous distinguons deux principaux types de transformations : celles qui augmentent le nombre de sommets d'un graphe, soit par duplication ou par dépliage, et celles qui définissent de nouvelles relations d'arcs à partir des relations existantes.

Copies, dépliages et structure arborescente. On énumère tout d'abord quelques transformations externes capables de produire un graphe avec un support plus grand. Dans les paragraphes qui suivent, nous fixons un graphe G à étiquettes dans Σ, et V_G le support de G.

K-copie. Soit K un alphabet fini disjoint de Σ, l'opération de K-copie appliquée à G duplique $|K|$ fois chaque sommet de G tout en conservant tous les arcs existant entre les sommets originaux, et ajoute un arc étiqueté par un élément différent k de K entre chaque sommet original de G et chacune de ses $|K|$ copies. Formellement, la K-copie G' de G est le graphe étiqueté sur $\Sigma \cup K$

$$\{u \xrightarrow{a} v \mid a \in \Sigma,\, s \xrightarrow[G]{a} t\} \cup \{u \xrightarrow{k} (u,k) \mid u \in V,\, k \in K\}.$$

Dépliage. Soit v un sommet de G, le dépliage de G par rapport à v est un arbre dont les branches représentent tous les chemins possibles dans G partant de v. Chaque nœud de cet arbre peut être associé à un « historique » de tous les sommets et arcs de G traversés au cours d'un certain parcours du graphe. Formellement, le dépliage de G par rapport au sommet v est l'arbre

$$\{wu \xrightarrow{a} wuau' \mid wu \in v(\Sigma V)^*,\, u \xrightarrow{a} u' \in G\}.$$

Les nœuds de cet arbre sont identifiés à des chemins d'origine v. Ces chemins peuvent être représentés par des séquences non vides de sommets de G (en commençant par v) et d'étiquettes d'arcs dans Σ alternées, soit l'ensemble $v(\Sigma V)^*$.

Structure arborescente. L'opération de structure arborescente est une variante d'une opération similaire appelée *itération arborescente* [Wal02]. Considérons l'ensemble V^+ des séquences non vides de sommets de G Soit $\#$ un nouveau symbole, on définit la structure arborescente de G comme le graphe $(\Sigma \cup \{\#\})$-étiqueté dont le support est V^+ et dont l'ensemble d'arcs est

$$\{wv \xrightarrow{\#} wvv \mid w \in V^+, v \in V\} \cup \{wu \xrightarrow{a} wv \mid u \xrightarrow{a} v \in G\}.$$

Interprétations du second ordre monadique. Une opération courante de logique appelée *interprétation* consiste à définir de nouvelles structures à l'aide de formules en s'appuyant sur les éléments d'une structure existante. Nous nous intéressons aux interprétations de graphes dans les graphes, et au fragment monadique de la logique du second ordre. Formellement, une *interprétation (du second ordre) monadique* de Γ dans Σ est une famille $I = (\phi_a(x,y))_{a\in\Gamma}$ de formules à deux variables libres du premier ordre de la logique MSO sur les graphes Σ-étiquetés. L'interprétation de G selon I est le Γ-graphe

$$I(G) = \{u \xrightarrow{a} v \mid G \models \phi_a(u,v)\}.$$

Remarquons que les interprétations monadiques ne permettent pas d'augmenter le nombre de sommets d'un graphe. Pour remédier à cela, on peut considérer des compositions de transformations augmentant la taille du support avec des interprétations monadiques. Dans le cas spécifique de la composition d'une K-copie et d'une interprétation monadique, on parle de *transduction (du second ordre) monadique*. Ces transformations très expressives sont présentées en détail dans [Cou94] dans le cadre plus général des structures relationnelles d'arité quelconque.

Marquages et restrictions. Même dans le cas où le nommage des sommets d'un graphe est inconnu, il peut être intéressant de sélectionner un certain sous-ensemble du support d'un graphe pour un usage ultérieur. Cette sélection peut bien sûr dépendre de la structure du graphe.

Il existe plusieurs façons de sélectionner des sous-ensembles de sommets dans un graphe. Premièrement, étant donné un sommet $r \in V_G$ quelconque, il est possible de considérer l'ensemble $X_{(r,L)}$ de tous les sommets accessibles dans G par un chemin d'origine r étiqueté par un certain mot du langage régulier $L \subseteq \Sigma^*$. Une autre façon de sélectionner des sommets de G est, étant donné une formule MSO $\phi(x)$ à une variable libre de premier ordre x, de considérer l'ensemble $X_\phi = \{v \in V \mid G \models \phi(v)\}$ des sommets satisfaisant ϕ. Dans le premier cas, on parle d'ensembles de sommets définis rationnellement, et dans le second cas d'ensembles MSO-définissables. Dans le cas où le sommet r peut être lui-même sélectionné à l'aide d'une formule MSO, les sélections rationnelles sont des cas particuliers de sélections monadiques.

Deux opérations courantes sur les graphes exploitant ces sélections de sommets sont les marquages et les restrictions. Formellement, le *marquage* d'un graphe G selon un ensemble de sommets X est le graphe

$$M_{\#}(G,X) = \{u \xrightarrow{a} v \mid u \xrightarrow[G]{a} v\} \cup \{u \xrightarrow{\#} u \mid u \in X\}$$

obtenu en ajoutant à G une boucle étiquetée par $\#$ à chaque sommet appartenant à X, où $\# \notin \Sigma$ est un nouveau symbole. Nous rappelons que la restriction d'un graphe G par rapport à un ensemble de sommets X est simplement le graphe $\{u \xrightarrow{a} v \mid u,v \in X,\ u \xrightarrow[G]{a} v\}$.

Les marquages et restrictions par rapport à un ensemble de sommets défini rationnellement ou définis par MSO sont respectivement appelés rationnels et MSO-définissables. Si l'ensemble de sommets X est MSO-définissable, ces deux opérations sont des cas particuliers d'interprétation monadique. Un cas particulier de restriction, appelé *restriction par accessibilité*, apparaît lorsque X est l'ensemble de tous les sommets accessibles depuis un certain sommet r.

Substitutions inverses. Une autre sous-classe d'interprétations du second ordre monadique, appelées substitutions rationnelles (ou finies) inverses, définit les arcs d'un nouveau graphe par rapport aux chemins existant dans le graphe original. Une substitution est une application h de Γ dans Σ^*, où Γ est un alphabet. Elle est appelée rationnelle (resp. finie) si l'image de chaque lettre de Γ par h est un langage rationnel (resp. un langage fini). L'application inverse de h à G est le Γ-graphe

$$h^{-1}(G) = \{u \xrightarrow{a} v \mid \exists w \in h(a), u \xrightarrow[G]{w} v\}.$$

Lorsque $h(a) = \varepsilon^* a \varepsilon^*$ pour tout a, on parlera plutôt de $h^{-1}(G)$ comme de l'*ε-clôture* de G. Cette opération appliquée au graphe des configurations d'une machine et suivie par une restriction à l'ensemble des sommets externes (qui est une restriction MSO-définissable), constitue une caractérisation externe du graphe de transition de la machine.

Composer et répéter des transformations. La particularité et le principal intérêt de toutes les transformations ci-dessus est qu'elles préservent la décidabilité de la théorie au second ordre monadique. Plus précisément, étant donné un graphe G de théorie monadique décidable, l'image de G par n'importe laquelle de ces transformations a toujours une théorie monadique décidable. Partant d'une formule MSO close sur le graphe image, il est en fait possible de construire une formule sur le graphe original possédant la même valeur de satisfaisabilité.

Cette observation produit directement une méthode pour caractériser des familles de graphes de théorie monadique décidable. Soit F une famille de graphes,

il est possible de caractériser de nouvelles familles par application simple de ces transformations, ou par composition ou itération de celles-ci. Le fait que ces transformations préservent la décidabilité de la théorie monadique implique que les nouvelles familles obtenues ont une théorie monadique décidable pourvu que ce soit le cas de F.

2.1.2.2 Systèmes d'équations et grammaires de graphes

Tous les langages de la hiérarchie de Chomsky, ainsi que les langages réguliers de termes par exemple, peuvent être définis à l'aide de grammaires. Les grammaires peuvent être vues comme des systèmes d'équations sur des ensembles appropriés d'opérateurs, dont les plus petites solutions sont les langages qu'elles engendrent. Il est possible de définir selon le même principe des grammaires sur les graphes. Les jeux d'opérateurs sur les graphes les plus utilisés sont les jeux d'opérateurs de remplacement d'hyperarcs (HR) et de remplacement de sommets (VR), tous deux ayant pour domaine les graphes coloriés (voir par exemple [Cou97] pour une présentation détaillée).

Opérateurs HR et grammaires HR. Les opérateurs HR consistent en l'ensemble des graphes coloriés finis, l'union disjointe, le re-coloriage de sommets et la fusion des sommets de même couleur. Les systèmes d'équations sur ces opérateurs, appelés systèmes HR, peuvent être représentées de façon naturelle sous la forme de grammaires à remplacement d'hyperarcs. Nous donnons seulement une définition informelle de ces grammaires.

Les hyperarcs peuvent être vus comme des flèches reliant plusieurs sommets d'un graphe (ou plus précisément dans ce cas d'un *hypergraphe*), et définis formellement comme des n-uplets $(a,v_1 \dots v_n)$, où a est l'étiquette de l'hyperarc et chaque v_i, $i \in [1,n]$ est un sommet. Le nombre n est appelé *arité* de l'hyperarc. Les ensembles d'hyperarcs dans un graphes sont définis comme des relations d'arité quelconque entre sommets. Une grammaire à remplacement d'hyperarcs est caractérisée par un ensemble d'étiquettes d'hyperarcs appelées non-terminaux, et par un ensemble de productions de la forme $A \to H$, où H est un hypergraphe colorié et A un non-terminal. Si A est d'arité n, on impose qu'exactement un sommet de H soit colorié par chaque couleur i dans $[1,n]$.

De façon informelle, une étape de dérivation d'une telle grammaire est définie en remplaçant chaque hyperarc étiqueté par un non-terminal A ayant une occurrence dans un hypergraphe G par la partie droite correspondante H dans les règles de production de la grammaire. Le remplacement d'un hyperarcs par un hypergraphe colorié est effectuée en prenant l'union disjointe de G et H, en fusionnant le i-ème sommet de G et l'unique sommet de H de couleur i pour chaque $i \in [1,n]$ et en retirant l'hyperarc A ainsi que toutes les couleurs des som-

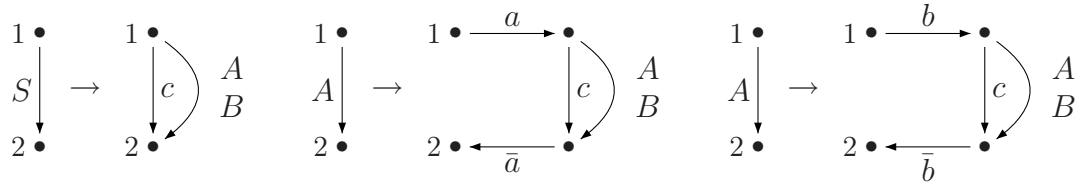


FIG. 2.3 – *Grammaire de graphes à remplacement d'hyperarcs.*

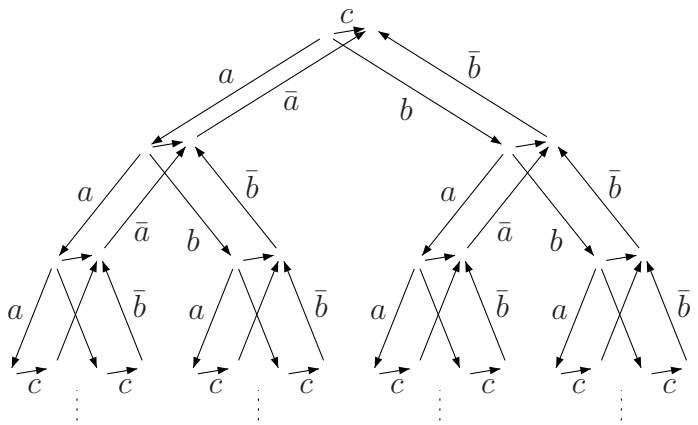


FIG. 2.4 – *Graphe infini engendré par la grammaire de la figure 2.3.*

---

mets du graphe obtenu. Le remplacement successif de chaque hyperarc forme une série d'hypergraphes finis. Le graphe (infini) engendré par la grammaire est la limite de cette série (qui ne contient plus aucun hyperarc). La figure 2.3 montre un exemple de grammaire HR. Le graphe engendré est représenté à la figure 2.4.

Opérateurs VR. Les opérateurs VR permettent l'union disjointe de graphes, l'ajout de sommets isolés coloriés, l'ajout d'arcs entre touts les couples de sommets de même couleur et le recoloriage des sommets à l'instar des opérateurs HR. Les graphes solutions des systèmes d'équations sur les opérateurs VR peuvent aussi être caractérisés à l'aide de grammaires dites « à remplacement de sommets », dont les règles de production spécifient les remplacement possibles de sommets non-terminaux par des graphes finis. Ceci implique de préciser de quelle façon les anciens arcs adjacents au sommet remplacé sont reconnectés au graphe fini qui le remplace. Nous ne donnerons pas plus de détail concernant cette famille dans ce document.

Extensions. D'autres familles étendant les ensembles d'opérateurs HR et VR ont été considérés. Dans [Col04], l'auteur propose plusieurs opérations nouvelles pour étendre l'ensemble VR, en particulier les produits synchrone et asynchrone de graphes, et obtient respectivement les familles VRS et VRA, et fournit une caractérisation équationnelle de plusieurs importantes familles de graphes généralisant les graphes VR-équationnels (cf. § 2.2).

2.1.2.3 Autres caractérisations externes

De la même façon que les ensembles réguliers de mots correspondent à l'ensemble des modèles des formules closes de logique du second ordre monadique, on peut définir des ensembles de graphes comme les modèles des formules closes de toute logique sur les graphes. Étant donnée une logique L, on parle d'ensembles L-définissables. Puisque les variables dans les formules ne font pas explicitement référence au nommage des sommets, il s'agit bien d'une caractérisation externe.

Enfin, nous pouvons évoquer la caractérisation de familles de graphes par des critères purement géométriques. L'exemple le plus connu concerne la famille des graphes à pile (*pushdown graphs*), qui sont caractérisés comme les graphes de degré fini possédant un nombre fini de décompositions par distance à partir d'un sommet quelconque (cf. § 2.2.1.2).

2.2 Familles de graphes infinis

Nous avons vu dans la première partie de ce chapitre plusieurs possibilités pour définir des graphes infinis de présentation finie. Nous avons aussi mentionné plus tôt certaines des raisons pour lesquelles il est pertinent de ne considérer de tels graphes qu'à isomorphisme près : on s'intéresse en fait aux propriétés structurelles d'un graphe, indépendamment du nommage particulier donné à ses sommets. Nous illustrons à présent ces idées sur une sélection des familles de graphes infinis les plus étudiées.

2.2.1 Graphes infinis de théorie monadique décidable

La logique du second ordre monadique est une logique très utile sur les graphes, combinant un bon pouvoir expressif (existence de chemins, confluence, cyclicité, etc.) avec un problème de satisfaisabilité décidable sur de nombreuses structures logiques importantes (demi-droite des entiers, arbres complets). Nous verrons dans ce paragraphe que ces résultats peuvent être étendus à de nombreuses autres familles de graphes, notamment en utilisant les transformations de graphes vues précédemment.

2.2.1.1 Premiers résultats

Büchi [Büc62] a montré que la théorie au second ordre monadique de l'ensemble des entiers muni de l'opérateur « successeur », ($\mathbb{N}$,S), est décidable. Ce résultat peut être interprété en termes de graphes comme la décidabilité de la logique MSO sur la demi-droite des entiers, qui est un graphe infini isomorphe à ($\mathbb{N}$,S). Ce graphe a pour ensemble de sommets les entiers naturels, et possède un arc entre tous les couples de sommets représentant des entiers consécutifs. La logique du second ordre monadique sur cette structure est parfois appelée $S1S$, ou logique monadique du second ordre pour la relation successeur.

Ce résultat fut plus tard étendu à un autre graphe infini par Rabin [Rab69], qui montra que la théorie monadique de l'arbre binaire complet infini Δ_2 est elle aussi décidable. Comme dans le cas précédent, on surnomme parfois cette logique $S2S$ pour une raison évidente. On fait en général référence à ce résultat comme le *théorème de Rabin pour les arbres*. De nombreux résultats concernant la décidabilité de MSO sur des structures plus générales sont démontrés par réduction à ce théorème.

2.2.1.2 Graphes d'automates à pile

Le vrai commencement d'une étude des graphes infinis pour eux-mêmes fut marqué par les travaux de Muller et Schupp. Dans [MS85], ils caractérisent l'ensemble des graphes d'automates à pile, dont les sommets sont les configurations accessibles depuis l'état initial d'un automate à pile, et dont les arcs représentent les transitions. Selon la terminologie que nous avons adoptée, les graphes d'automates à pile sont les restrictions par accessibilité depuis la configuration initiale des graphes de configuration d'automates à pile.

Le principal résultat concernant cette famille de graphes est qu'ils ont une théorie monadique décidable. Ce résultat fut obtenu à l'aide de la caractérisation géométrique suivante. Soit G un graphe, et v_0 l'un quelconque de ses sommets. On note $G_{\geq n}$ l a restriction de G à l'ensemble de ses sommets situés à distance au moins n de v_0 (ou la distance entre deux sommets est prise comme la longueur du plus court chemin dirigé reliant le premier au second). Muller et Schupp ont démontré qu'un graphe est isomorphe à un graphe d'automate à pile si et seulement si l'ensemble des composantes connexes de la famille $\{G_{v_0,n} \mid n \geq 0\}$ est fini à isomorphisme près quel que soit v_0 (voir la figure 2.5 pour une illustration cette idée[1]). Ce résultat leur a ensuite permis, en utilisant le théorème de Rabin, de conclure que les graphes d'automates à pile ont une théorie monadique décidable.

Enfin, il est relativement direct de montrer que les langages de tels graphes entre des ensembles initiaux et finaux finis sont les langages algébriques.

1. Merci à Thomas Lavergne pour m'avoir prêté ce dessin.

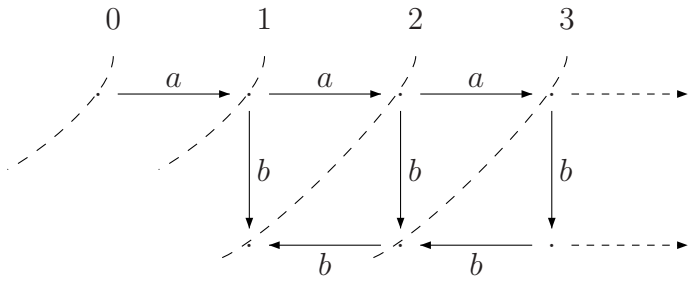


FIG. 2.5 – *Décomposition d'un graphe d'automate à pile par distance depuis sa racine.*

2.2.1.3 Graphes réguliers ou équationnels

Les graphes engendrés par les grammaires à remplacement d'hyperarcs sont parfois appelés réguliers, ou équationnels. Nous les appellerons graphes HR-équationnels pour éviter toute ambiguïté. Courcelle a défini et étudié cette famille, et a démontré en particulier que leur théorie monadique est décidable ([Cou90]).

Il a été montré dans [Cau92] que la famille des graphes HR-équationnels de degré fini possédant une racine coïncident à la fois avec les graphes d'automates à pile et avec les restrictions par accessibilité des graphes de réécriture préfixe des systèmes de réécriture finis de mots (autre nom pour la clôture à droite des graphes de réécriture de systèmes finis de mots).

Si l'on omet cette restriction par accessibilité, on obtient malgré tout une intéressante équivalence entre les graphes HR-équationnels de degré fini, les restrictions rationnelles des clôtures à droite de systèmes de réécriture finis de mots, les restrictions rationnelles des graphes des configurations d'automates à pile[2] ([Cau95]) et les clôtures rationnelles à droite des relations finies de mots ([Cau96]).

2.2.1.4 Graphes préfixe-reconnaissables

Les graphes préfixe-reconnaissables ont été définis dans [Cau96]. Ils sont un bon exemple de famille de graphes infinis possédant un grand nombre de caractérisations internes et externes différentes, comme le résume le théorème d'équivalence suivant (où seules les présentations les plus significatives sont mentionnées).

Théoreme 2.7. *Soit G un graphe étiqueté. Les propositions suivantes sont équivalentes :*

1. *G est isomorphe à la clôture rationnelle à droite d'un graphe reconnaissable (ou, de façon équivalente, au graphe des réécritures préfixes d'un système reconnaissable de mots),*

2. Nous insistons sur la nuance entre les graphes d'automates à pile de Muller et Schupp, qui sont restreints aux sommets accessibles, et les graphes des configurations d'automates à pile.

2. *G est isomorphe au graphe de transition d'un automate à pile [Sti98],*
3. *G est isomorphe au graphe de type Cayley d'un système de réécriture de mots avec chevauchements préfixes [CK02a],*
4. *G peut être construit à partir de l'arbre binaire complet par un marquage rationnel suivi d'une substitution rationnelle inverse,*
5. *G peut être construit à partir de l'arbre binaire complet par une interprétation monadique [Blu01],*
6. *G est VR-équationnel [Bar97].*

Chacune de ces caractérisations est attribuée à son auteur. Plusieurs propriétés de cette famille découle de cette variété de présentations. La décidabilité de MSO sur les graphes préfixe-reconnaissables est une conséquence directe de leurs présentations externes par transformations MSO-compatibles de l'arbre binaire complet (points (4) et (5)). Le fait que les traces des graphes préfixe-reconnaissables sont les langages hors-contexte était déjà connu dans [Cau96], mais peut être également vu comme une conséquence de (2). Enfin, le fait que cette famille soit close par restriction rationnelle et substitution rationnelle inverse est une conséquence de (1).

D'autres conséquences de ces résultats ont trait aux sous-familles des graphes préfixe-reconnaissables. Par exemple, la décidabilité de la théorie monadique des graphes VR-équationnels peut être déduite directement de (6). Il est de plus relativement clair que les graphes HR-équationnels sont une sous-famille des VR-équationnels. (En fait, il a été montré dans [Bar98] que les graphes HR-équationnels sont précisément les graphes VR-équationnels de largeur arborescente bornée[3].) Comme les graphes HR-équationnels et les graphes d'automates à pile sont tous des graphes préfixe-reconnaissables ([Cau96]), on obtient également une nouvelle démonstration de la décidabilité de leurs théories monadiques respectives.

Notons enfin qu'en raison du fait que l'équivalence de traces est une forme d'équivalence plus large que l'isomorphisme, il peut exister plusieurs familles de graphes distinctes dont les traces sont la même famille de langages. Dans le cas présent, comme les traces des graphes d'automates à pile comme des préfixe-reconnaissables sont les langages hors-contexte, toute famille intermédiaire (par exemple les graphes HR-équationnels) ont la même famille de traces (voir [Sti98] pour plus de détails).

2.2.1.5 La hiérarchie C.

Dans le même esprit que les graphes préfixe-reconnaissables, une hiérarchie infinie et stricte de familles de graphes obtenues par itération de transformations

3. La largeur arborescente d'un graphe est en quelque sorte une mesure de sa ressemblance à un arbre. Sa définition n'est pas triviale et non pertinente pour ce travail.

sur les graphes a été étudiée dans [Cau02]. Le niveau 0 de cette hiérarchie consiste en l'ensemble des graphes finis. Les graphes du niveau $n > 0$ sont définis comme les dépliages suivis de substitutions rationnelles inverses des graphes de niveau $n-1$. Pour référence ultérieure[4], nous appellerons cette hiérarchie la « Hiérarchie C. ». Une conséquence directe de cette définition est que tous les graphes de la hiérarchie C. ont une théorie monadique décidable. Des études plus approfondies de cette hiérarchie et des comparaisons avec d'autres familles sont à l'origine du résultat d'équivalence suivant.

Théoreme 2.8. *Soit G un graphe étiqueté. Les propositions suivantes sont équivalentes :*

1. *G appartient au niveau n de la hiérarchie C. (c'est à dire est défini comme l'application successive de n dépliages composés avec des substitutions rationnelles inverses),*
2. *G est défini comme l'application successive de n opérations de structure arborescente composés avec des transductions monadiques ([CW03]),*
3. *G est isomorphe au graphe de transition d'un automate à pile d'ordre supérieur de niveau n ([CW03]),*
4. *G est isomorphe au graphe d'une relation « préfixe-reconnaissable d'ordre supérieur » ([Car05]).*

De plus, Caucal a montré dans [Cau02] que les arbres obtenus par dépliage des graphes de la hiérarchie incluent à la fois les solutions des schémas de programmes d'ordre supérieur *sûrs* [KNU02] et une hiérarchie de termes définis par une certaine forme de substitution de premier ordre itérée [CK02b].

En tant que graphes de transition des automates à pile d'ordre supérieur, les traces des graphes de cette hiérarchie coïncident avec la hiérarchie OI de langages (cf. § 1.3.4). Nous rappelons que ces langages sont une sous-famille des langages contextuels.

2.2.2 Graphes rationnels et graphes automatiques

La famille des graphes rationnels doit son nom au fait que les ensembles d'arcs de ces graphes sont donnés comme des relations rationnelles sur les mots, c'est à dire des relations reconnues par des transducteurs finis. Dans ce paragraphe, nous présentons la famille générale des graphes rationnels et certaines de ses sous-familles, en particulier les graphes rationnels synchronisés (ou automatiques) et synchrones.

Les graphes rationnels sont simplement définis comme les graphes de calcul d'ensembles finis de transducteurs de mots (cf. § 2.1.1.1 et § 1.4.2). Ainsi, tout Σ-graphe rationnel G est caractérisé par une famille $(T_a)_{a\in\Sigma}$ de transducteurs

4. (et pour une raison de modestie et de discrétion)

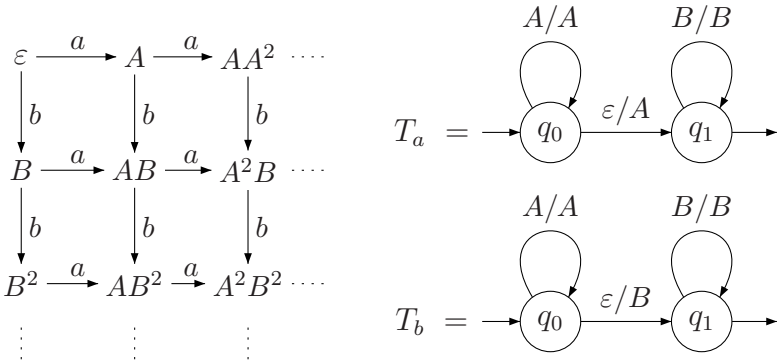


FIG. 2.6 – *La grille et les transducteurs la définissant.*

de mots sur un alphabet Γ quelconque. Nous rappelons que par définition des graphes de calcul, il existe un arc étiqueté par a dans G entre deux sommets u et $v \in \Gamma^*$ si $(u,v) \in T_a$.

Dans [Mor00], d'autres caractérisations de cette famille sont données. Il est montrée que les graphes rationnels coïncident avec les restrictions rationnelles d'une certaine sous-famille de substitutions linéaires inverses de l'arbre binaire complet.

La figure 2.6 montre un exemple de graphe rationnel, la *grille* bi-dimensionnelle infinie, ainsi que les transducteurs la définissant. Notons qu'il s'ensuit du lemme 1.29 que le support d'un graphe rationnel est un sous-ensemble rationnel de Γ^*.

Les graphes rationnels à transducteurs synchronisés furent déjà définis par Blumensath et Grädel in [BG00] sous le nom de graphes *automatiques*, et par Rispal sous le nom de graphes rationnels synchronisés. Par un léger abus de langage, on parlera de graphes synchrones pour se référer à des graphes rationnels définis par des transducteurs synchrones, et de graphes séquentiels synchrones dans le cas correspondant.

Il suit des définitions correspondantes (voir § 1.4.2) que les graphes séquentiels synchrones, synchrones, synchronisés et rationnels forment une hiérarchie croissante. Cette hiérarchie est stricte (à isomorphisme près) : premièrement, les graphes séquentiels sont déterministes, tandis que les graphes synchrones peuvent ne pas l'être. Deuxièmement, les graphes synchrones ont un degré fini, ce qui n'est pas nécessairement le cas des graphes synchronisés. Enfin, pour séparer les graphes synchronises des graphes rationnels, nous pouvons utiliser la propriété suivante relative au taux de croissance du degré sortant dans les graphes rationnels et synchronisés de degré fini.

Proposition 2.9. *[Mor01] Pour tout graphe rationnel G de degré sortant fini et*

tout sommet x, il existe $c \in \mathbb{N}$ tel que le degré sortant de tout sommet à distance n de x est au plus c^{c^n}.

Cette borne supérieure peut être atteinte : considérons le graphe rationnel non étiqueté $H = \{T\}$ où T est le transducteur sur $\Gamma = \{A,B\}$ à un état q_0 à la fois initial et final et une transition $q_0 \xrightarrow{X/YZ} q_0$ pour tous X,Y et $Z \in \Gamma$. Ce graphe a un degré sortant égal à $2^{2^{n+1}}$ à la distance n du sommet A. Dans le cas des graphes synchronisés de degré fini, la borne sur le degré sortant est simplement exponentielle.

Proposition 2.10. *[Ris02] Pour tout graphe synchronisé G de degré sortant fini et tout sommet x, il existe $c \in \mathbb{N}$ tel que le degré sortant de tout sommet à distance n de x est au plus c^n.*

Il découle de ces observations que le graphe H ci-dessus est rationnel mais pas synchronisé. Donc, les graphes synchronisés forment une sous-famille stricte des graphes rationnels. Les graphes rationnels ont une théorie monadique indécidable, même dans le cas synchrone. Les graphes synchronisés ont une théorie au premier ordre décidable, mais il existe des graphes rationnels dont ce n'est pas le cas. Une propriété très importante des graphes rationnels est que leurs langages de traces entre des ensembles rationnels (ou finis) de sommets initiaux et finaux sont les langages contextuels [MS01]. Ce résultat fut plus tard étendu aux graphes synchronisés [Ris02], et même aux graphes synchrones quand seuls des ensembles rationnels de sommets initiaux et finaux sont considérés. Au paragraphe 4.2, nous fournirons des preuves simplifiées de ces propriétés, et au paragraphe 4.4 nous établissons une comparaison des graphes rationnels avec une autre famille de graphes infinis dont les traces sont les langages contextuels.

2.2.3 Graphes de Turing

Les graphes de Turing, aussi appelés graphes calculables, sont les graphes de transition de machines de Turing. Ils ont été étudiés dans [Cau03], où l'auteur montre que ces graphes peuvent être caractérisés de plusieurs façons : ils coïncident avec les graphes de calcul de machines de Turing vues comme des transducteurs, avec les substitutions rationnelles inverses des graphes rationnels, avec les restrictions rationnelles des images de l'arbre binaire complet par toute famille suffisamment générale de substitutions inverses (ou « suffisamment général » est défini en référence à une sous-famille des langages hors-contexte), et avec les graphes de type Cayley des systèmes de réécriture non restreints de mots.

2.2.4 Graphes sur les termes

Comme elles admettent pour la plupart des caractérisations sous forme de graphes de réécriture de mots, les familles de graphes précédemment citées peu-

vent être étendues de façon naturelle à des graphes de réécriture de termes.

Les graphes d'automates à pile sont étendus aux termes en considérant les restrictions par accessibilité des graphes de réécriture de systèmes clos de termes. D'après [DHLT90], ces graphes ont une théorie du premier ordre avec accessibilité décidable. Ils furent aussi étudiés par Löding dans [Löd02]. En remplaçant la restriction par accessibilité par une restriction à un langage de termes reconnu par un automate descendant déterministe, on obtient une généralisation des graphes de réécriture préfixe de systèmes finis de mots, et même des graphes préfixe-reconnaissables quand l'ensemble de règles de réécriture considéré est reconnaissable. Enfin, les graphes terme-automatiques ont été considérés dans [BG00] en utilisant une notion de relations synchronisées de termes équivalente à celle du paragraphe 1.4.3.3. Toutes ces familles de graphes ont ensuite reçu des caractérisations équationnelles dans [Col04].

2.3 Graphes infinis et vérification

Un important défi de la vérification automatique d'aujourd'hui est d'étendre les méthodes existantes utilisant les automates finis (cf. § 1.6), dont le succès est maintenant largement reconnu, au cas des systèmes à ensembles infinis d'états. Ce besoin est dû au fait que les abstractions finies des systèmes que l'on désire étudier ne permettent pas en général de vérifier de propriétés très précises ou très subtiles des systèmes, dont plusieurs aspects sont sources d'infinitude, tant du point de vue des domaines de données considérés que des mécanismes de contrôle.

2.3.1 *Model Checking* symbolique

Le problème de la vérification de modèles consiste à déterminer se un certain système vérifie une propriété donnée exprimée dans une logique temporelle ou une logique classique. Parmi les propriétés intéressantes les plus simples se trouvent les propriétés de *sûreté*, qui concernent la relation d'accessibilité d'un système. Une instance commune de ce genre de propriétés peut être formulé par la question *Est-il possible d'atteindre une configuration de l'ensemble B depuis toute configuration de l'ensemble A ?* où A et B sont deux ensembles de configurations du système considéré.

Pour répondre à ce genre de questions sur des systèmes infinis, un pré-requis indispensable est de pouvoir représenter des ensembles potentiellement infinis de configurations de façon finie, et de calculer leurs images par la relation de transition du système, ou mieux encore par la clôture transitive de cette relation. La représentation choisie pour ces ensembles doit aussi être close par un certain nombre d'opérations utiles comme l'union et l'intersection. Ce cadre de raisonnement est en général appelé *vérification symbolique*.

Un choix très courant pour la représentation symbolique d'ensembles de configurations codées comme des mots est l'utilisation de langages réguliers, qui possèdent de nombreuses propriétés de fermeture et propriétés algorithmiques intéressantes. Dans ce cas, on parlera plus volontiers de *model checking régulier*. Récemment, de nombreux travaux ont montré que les automates finis sont un moyen de représentation adapté pour les ensembles de configurations, et permettent notamment de traiter de manière uniforme une grande variété de systèmes dont les systèmes à pile, les systèmes à canaux de communication FIFO, les réseaux paramétrés de processus, les systèmes à compteurs, etc. [BEM97, BGWW97, KMM+97, ABJ98, WB98, BJNT00, Bou01, HJJ+95]. Dans tous les cas, une propriété clé est la *préservation de la régularité* des ensembles de configurations par la relation de transition du système (ou sa clôture transitive). On dit qu'une relation préserve la régularité si l'image de tout ensemble régulier par cette relation est elle aussi régulière. Bien sûr, toutes ces idées peuvent être également envisagées dans le cas des termes, puisque les ensembles réguliers de termes partagent la plupart des bonnes propriétés des ensembles réguliers de mots. On peut trouver des travaux dans cette direction par exemple dans [AJMD02, BT02].

De nombreux modèles suffisamment expressifs pour représenter des classes intéressantes de programmes et de systèmes engendrent immédiatement des problèmes de vérification indécidables pour la plupart des propriétés non triviales. Par exemple, si l'on dispose de règles d'évolution suffisamment générales, il est très simple de coder les calculs d'une machine de Turing grâce aux transitions d'un réseau paramétré linéaire, ce qui rend le problème de l'accessibilité indécidable dans un tel modèle.

Il existe deux approches principales pour faire face à cette observation négative. La première est de considérer des algorithmes approchés, ou des procédures de demi-décision, pour donner une réponse partielle à des problèmes autrement insolubles. La seconde option, plus proche de nos préoccupations, est de réduire l'expressivité du modèle original et de caractériser des classes de systèmes pour lesquels les problèmes de vérification sus-mentionnés sont décidables. Dans le reste de ce paragraphe, nous allons évoquer deux sujets actifs de la communauté de vérification de systèmes infinis qui admettent une reformulation directe en termes de graphes infinis.

2.3.2 Vérification de programmes récursifs.

La classe de programmes la plus évidente pour laquelle, même après l'habituelle abstraction des variables vers des domaines finis, un ensemble infini de configurations doit être considéré est celle des programmes contenant des procédures récursives. De tels programmes sont traditionnellement modélisés en utilisant une traduction naturelle vers les systèmes à pile (c'est à dire les automates

à pile sans entrée). La vérification de propriétés dans ce cadre peut ensuite directement être vue comme un problème de model-checking sur des graphes des configurations ou de transition de systèmes à pile. Les systèmes à pile et leurs problèmes de décision et d'analyse associés (analyse d'accessibilité, model checking, jeux, synthèse de contrôleurs, etc.) ont été largement étudiés au cours des dernières années [Cau92, BCS96, Wal96, BEM97, EHRS00, Cac02, AEM04].

De nouveaux résultats intéressants concernant les automates à pile d'ordre supérieur et la décidabilité de la théorie monadique de leurs graphes de transition peuvent avoir des conséquences intéressantes en termes de vérification de programmes. En effet, au même titre que les automates à pile sont parfaitement adaptés à la modélisation de programmes séquentiels récursifs, les automates à pile d'ordre supérieur peuvent permettre de modéliser des *programmes fonctionnels d'ordre supérieur*, c'est à dire des programmes récursifs dont les fonctions peuvent prendre en paramètre ou rendre comme résultat d'autres fonctions. Par conséquent, il est naturel de se demander si les techniques existantes peuvent s'étendre à ce nouveau cas. Le chapitre 5 présente un premier pas vers un algorithme d'analyse d'accessibilité symbolique adapté de [BEM97] pour une sous-famille des systèmes à pile d'ordre supérieur.

2.3.3 Systèmes paramétrés et dynamiques

Une autre classe de modèles, désignés comme des systèmes paramétrés, visent à vérifier des programmes ou des systèmes indépendamment de la valeur donnée à un certain paramètre, voire de synthétiser la ou les valeurs du paramètre pour lesquelles la propriété recherchée est vérifiée. Pour chaque instance du paramètre en question, le modèle est fini, mais il est potentiellement nécessaire de tester un nombre infini de cas de taille arbitraire.

On peut par exemple considérer des réseaux de processus communiquants de topologie fixée s'envoyant et recevant des messages par l'intermédiaire de canaux de communication. La topologie du réseau considéré peut être linéaire, circulaire, en étoile, ou même quelconque dans le pire des cas. Le paramètre en question est la topologie initiale du réseau (en particulier le nombre de processus), et l'état initial de chacun des processus. Ceux-ci peuvent évoluer en changeant d'état de concert avec leurs processus voisins. Le propriétés que l'on peut être amené à vérifier sont la vivacité du réseau (c'est à dire l'absence d'inter-blocages), l'accès possible à une configuration acceptante (ou d'erreur), ou des propriétés plus générales exprimées dans une logique quelconque. Les réseaux linéaires (ou circulaires) et arborescents sont facilement représentés par des mots ou des termes, où chaque symbole représente une abstraction finie de l'état courant de chaque processus du réseau. Les réseaux de topologie plus générale sont eux représentés par des graphes coloriés. Les règles de transition du système peuvent être modélisées par

des règles de réécriture préservant la taille (ou la structure dans le cas des termes et des graphes).

Des problèmes plus difficiles voient le jour quant on s'intéresse à des réseaux de processus dont la topologie peut évoluer au cours même du calcul, par exemple si des processus peuvent se terminer ou apparaître, des canaux de communications s'ouvrir ou se fermer, etc. Les transitions sont dans ce cas représentées par des règles de réécriture quelconques. Le chapitre 3 cherche à caractériser des classes de systèmes de réécriture de termes dont la relation de dérivation peut être calculée et représentée de façon finie, ce qui donne lieu à des applications potentielles dans ce dernier cadre.

Chapitre 3

Systèmes de réécriture de termes à dérivation rationnelle

L'un des arguments du paragraphe 2.1.1.2 est que les systèmes de réécriture fournissent un cadre très naturel à la représentation interne de familles de graphes infinis. Des systèmes de réécriture d'expressivité équivalente à tous les principaux types d'accepteurs de mots sont très faciles à définir tout en fournissant une meilleure flexibilité, en particulier pour la modélisation de diverses classes de systèmes pour la vérification automatique, comme des réseaux paramétrés de processus communiquants (cf. § 2.3). Dans ce contexte, il est intéressant de pouvoir représenter la relation d'accessibilité d'un système donné, ou à défaut l'image d'un ensemble régulier par dérivation selon un tel système.

Le but de ce chapitre est d'étendre certaines approches sur les systèmes de réécriture de mots au cas des termes, en mettant l'accent sur le calcul des relations de dérivations. Dans [Cau00], une classification des systèmes de réécriture de mots s'appuyant sur une condition syntaxique précise sur les règles de réécriture est menée. L'objectif est d'identifier des classes de systèmes dont la dérivation est rationnelle, c'est à dire acceptée par un transducteur fini, et de construire ce transducteur le cas échéant. Il est ressorti de cette étude la définition de quatre classes de systèmes, les systèmes gauches, droits, préfixes et suffixes, dont les relations de dérivations sont montrées rationnelles et effectivement calculables. Mieux encore, dans le cas des systèmes gauches ou droits, ces relations coïncident précisément avec la famille des relations rationnelles : les règles gauches et droites peuvent être vues comme une forme plus générale de transitions de transducteurs finis.

Le critère de classification utilisé dans [Cau00] peut facilement être adapté aux termes, comme on le verra au paragraphe 3.2 ci-dessous, et des extensions des familles de systèmes gauches, droits, préfixes et suffixes peuvent être définies. Cependant, pour pouvoir étendre de façon pertinente les résultats précédemment

cités aux termes, il est nécessaire de se munir d'une notion appropriée de relations binaires sur les termes possédant une représentation finie et un test d'appartenance décidable. Il s'avère que les transducteurs de termes habituels ne sont pas suffisamment expressifs pour capturer les dérivations de tous les systèmes que nous envisageons. Pour cette raison, nous avons choisi d'utiliser la classe de relations rationnelles de termes définie dans [Rao97], dont la définition est rappelée au paragraphe 3.1. Enfin, une présentation détaillée des résultats obtenus est donnée dans les parties restantes de ce chapitre. On démontrera que l'une des classes de systèmes considérées a des relations de dérivation non récursives, tandis que celles des trois autres classes peuvent être représentées de façon effectives comme des relations rationnelles au sens de [Rao97].

3.1 Relations rationnelles de termes

Comme évoqué au paragraphe 1.4.3.4, plusieurs tentatives d'extension des relations rationnelles des mots aux termes ont été faites, et de nombreuses notions différentes ont vu le jour. Dans ce paragraphe, nous avons rappelé certaines des caractéristiques des relations rationnelles de mots, et les avons établi comme liste de critères pour déterminer l'adéquation d'une extension particulière par rapport au cas des mots.

Dans [Rao97], Raoult propose une notion de rationalité pour les n-uplets de termes par rapport à l'union, la substitution et la substitution itérée. On peut aussi voir cette notion comme la définition d'une classe de relations binaires sur les n-uplets de termes, et donc aussi en particulier de relations binaires de termes. Cette définition est justifiée par sa ressemblance au cas des mots sur plusieurs aspects : elle coïncide avec les relations rationnelles de mots quand elle est restreinte au domaine des mots (représentés comme des termes sur des symboles d'arité 1), elle a un test d'appartenance décidable, est fermée par union et par inverse, et enfin peut être caractérisée à l'aide d'au moins deux formalismes naturels équivalents, à savoir des expressions rationnelles et des grammaires de n-uplets de termes. La principale différence cependant entre ces relations et les relations rationnelles de mots est qu'elles ne sont pas fermées par composition.

Nous présentons un certain nombre de nouvelles notations avant de donner la définition des ensembles rationnels de n-uplets de termes.

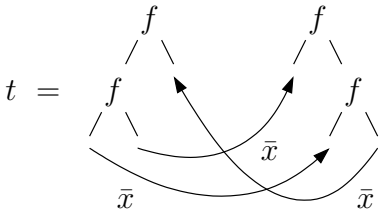
3.1.1 Multivariables

On généralise la notion de variable de terme afin de pouvoir lier plusieurs positions dans un mot de termes. Intuitivement, on associe à chaque variable une arité, avec l'idée qu'une variable d'arité n (appelée pour éviter toute confusion

multivariable) est associée à un n-uplet de positions dans un terme ou dans un mot de termes. Pour une écriture plus aisée, on décompose chaque multivariable d'arité n en une séquence de n variables simples *uniques*.

Formellement, soit F un alphabet gradué et X un ensemble de variables. On appelle *multivariable* tout mot $\bar{x}$ de X^* contenant au plus une occurrence de chaque variable le composant. Par exemple, $\bar{x} = x_1x_2x_3$ est une multivariable de longueur 3. Nous voulons être en mesure de considérer des mots de termes contenant plus d'une occurrence (ou *instance*) d'une multivariable donnée. Pour éviter de confondre des variables appartenant à des instances différentes de la même multivariable, et être sûr de pouvoir associer chaque variable à la bonne instance, il est nécessaire de distinguer explicitement chaque instance d'une même multivariable. Soit $\bar{x}$ une multivariable, l'ensemble des instances de $\bar{x}$ est $\{\bar{x}^i \mid i \in \mathbb{N}\}$. Si $\bar{x} = x_1 \dots x_n$, on notera l'instance x^i de x comme $x^i = x_1^i \dots x_n^i$. Si l'on se représente des mots de termes comme des forêts, on peut voir une multivariable d'arité n comme une flèche reliant n feuilles distinctes (un hyperarc).

Exemple 3.1. Soit $\bar{x} = x_1x_2$ une multivariable d'arité 2. Le mot de termes linéaire $t = f(f(x_1^1,x_1^3),x_2^2)\ f(x_2^3,f(x_2^1,x_1^2))$ contient trois instances de $\bar{x}$, notées respectivement $x_1^1x_2^1$, $x_1^2x_2^2$ et $x_1^3x_2^3$. On peut représenter t graphiquement comme suit :



Soit $\overline{X} \subseteq X^*$ un ensemble de multivariables. On suppose que, pour tous $\bar{x} = x_1 \dots x_n, \bar{y} = y_1 \dots y_m \in \overline{X}$, toutes les variables x_i et y_j de $\bar{x}$ et $\bar{y}$ sont distinctes. Dans ce chapitre, on considérera uniquement des mots de termes linéaires sur F dont les variables appartiennent toutes à une et une seule instance complète d'une multivariable (où une instance de multivariable est complète si aucune de ses variables ne manque)[1]. L'ensemble de tels mots de termes est noté $T^*(F,\overline{X})$. Quand seulement une instance $\bar{x}^1$ de $\bar{x}$ apparaît dans un mot de termes, on écrit simplement $\bar{x}$. Deux mots de termes sont considérés comme égaux s'ils ne diffèrent que par la numérotation de leurs instances de multivariables.

Exemple 3.2. Soit $\bar{x} = x_1x_2x_3$ une multivariable de longueur 3. Le terme fx_1x_3 n'appartient pas à $T(F,\overline{X})$ parce qu'il contient une instance incomplète de $\bar{x}$.

L'opération de substitution est elle aussi étendue pour prendre en compte les multivariables. Nous abrégeons la substitution $\{\bar{x}_j^i \mapsto s_j \mid j \in [1,n]\}$ en $\{\bar{x}^i \mapsto s\}$, où $\bar{x}^i = \bar{x}_1^i \dots \bar{x}_n^i$ est une instance de la multivariable $\bar{x}$ et $s = s_1 \dots s_n$ est un mot

1. Dans ce cadre, le nombre total de variables distinctes apparaissant dans un terme n'est pas borné, mais est cependant fini

de termes. Comme on ne considère que des mots de termes linéaires, lorsqu'on effectue une substitution $t\{\bar{x}^i \mapsto s\}$ on suppose également que la numérotation de toute instance de multivariable présente dans s est choisie plus grande que celles de toutes les instances de la même multivariable déjà présentes dans t.

Exemple 3.3. Soit $\bar{x} = x_1x_2$ une multivariable, $\sigma = \{\bar{x}^1 \mapsto ab,\ \bar{x}^2 \mapsto cd\}$ une substitution et $t = f(f(x_1^1,x_2^1),x_2^2)\ x_1^2$ un terme muni de deux instances de $\bar{x}$, on a $t\sigma = f(f(a,b),d)\ c$.

3.1.2 Ensembles rationnels de n-uplets de termes

Nous donnons à présent la définition d'une opération de produit sur les mots de termes, dont on tire une notion de rationalité sur ce domaine. Ce produit est une extension de l'opération de substitution habituelle sur les termes linéaires. Soit t un mot de termes dans $T^*(F,\overline{X})$ (cf. paragraphe précédent), $\bar{x}$ une multivariable d'arité n avec $k \geq 0$ instances $\bar{x}^1 \dots \bar{x}^k$ dans t (soit un total de $n*k$ variables) et M un ensemble de mots de termes de longueur n. On définit $t \cdot_{\bar{x}} M$ comme l'ensemble de mots de termes obtenus en remplaçant chaque instance de $\bar{x}$ dans t par un mot de terme (potentiellement à chaque fois différent) de M. Formellement,

$$t \cdot_{\bar{x}} M = \left\{ t\{\bar{x}^1 \mapsto s_1, \dots \bar{x}^k \mapsto s_k\} \mid \forall i \in [1,k], s_i \in M \right\}$$

où k est le nombre d'occurrences de $\bar{x}$ dans t. Cette opération est étendue par extension additive aux ensembles de la façon habituelle : $L \cdot_{\bar{x}} M = \{t \cdot_{\bar{x}} M \mid t \in L\}$. Nous insistons sur le fait que la longueur d'une multivariable $\bar{x}$ et de tous les mots de termes de M doivent être égales pour que ce produit soit défini.

Si L est un mot de termes de longueur n et $\bar{x}$ est toujours une multivariable de longueur n, on définit $L^{0_{\bar{x}}} = \{\bar{x}\}$ et $L^{n_{\bar{x}}} = \{\bar{x}\} \ \cup \ L \cdot_{\bar{x}} L^{n-1_{\bar{x}}}$. L'étoile de L sur la multivariable x est donc comme d'habitude $L^{*_{\bar{x}}} = \bigcup_{n \geq 0} L^{n_{\bar{x}}}$. On peut maintenant définir la notion de rationalité associée à cette opération de produit.

Remarque 3.4. Il est important de comprendre que, de par la forme de la définition de cette opération de concaténation, des instances différentes d'une même multivariable peuvent être substituées par des mots de termes différents lors d'un produit. Par exemple, soit $F = \{f^{(2)},a^{(0)}\}$, l'expression rationnelle $(fx^1x^2)^{*_x} \cdot_x a$ décrit l'ensemble $T(F)$ tout entier et pas, comme on pourrait le croire, l'ensemble des termes équilibrés sur F.

Définition 3.5 ([Rao97])**.** La famille Rat_n des ensembles rationnels de n-uplets de termes est la plus petite famille contenant les ensembles finis de n-uplets et fermée par les opérations suivantes :

1. $L \in Rat_n \ \wedge \ M \in Rat_n \ \Rightarrow \ L \cup M \in Rat_n$
2. $L \in Rat_n \ \wedge \ \bar{x} \in X^m \ \wedge \ M \in Rat_m \ \Rightarrow \ L \cdot_{\bar{x}} M \in Rat_n$
3. $L \in Rat_n \ \wedge \ \bar{x} \in X^n \ \Rightarrow \ L^{*_{\bar{x}}} \in Rat_n$

La famille *Rat* des ensembles rationnels de mots de termes est l'union de toutes les familles Rat_n, pour $n \geq 0$.

Notons que cette notion de rationalité diffère de celle qui est définie dans [LW01] par exemple, car la juxtaposition de deux mots de termes (ou « produit série » dans cet article) n'est pas directement pris en compte, et la substitution est effectuée simultanément sur plusieurs variables. De cette définition découle une notion immédiate d'*expression rationnelle*, qui étend la notion habituelle sur les mots. Remarquons aussi que l'ensemble Rat_1 ne coïncide pas avec l'ensemble des langages réguliers de termes, comme le montre l'exemple suivant.

Exemple 3.6. L'expression rationnelle $fxy \cdot_{xy} (gxgy)^{*_{xy}} aa$ sur l'alphabet $F = \{f^{(2)},g^{(1)},a^{(0)}\}$ représente le langage $\{fg^n ag^n a \mid n \geq 0\} \in Rat_1$, qui n'est pas un langage régulier de termes.

3.1.3 Grammaires de n-uplets de termes

Rappelons maintenant la notion de grammaires utilisée dans [Rao97] pour engendrer des ensembles de mots de termes.

Définition 3.7 (Grammaire à multivariables)**.** Une *grammaire à multivariables* (ou simplement grammaire dans ce chapitre) est un quadruplet $G = (F,N,S,P)$, où F est un alphabet gradué, N est un ensemble de multivariables distinctes appelées non-terminaux, $S \in N$ est le non-terminal initial aussi appelé axiome et P est un ensemble de productions. Par analogie avec les grammaires de mots, on note les non-terminaux à l'aide de lettres capitales. Les productions sont de la forme (A,t), aussi notées $A \to t$ avec $A = A_1 \dots A_n \in N$ et $t = t_1 \dots t_n \in T(F,N)^n$. On appelle A la partie gauche et t la partie droite de la règle.

Une étape de dérivation de G depuis le mot de termes s est définie comme la substitution d'une instance d'un non-terminal A dans s par la partie droite t d'une règle de partie gauche A. Formellement, on note $s \underset{G}{\longrightarrow} s'$ si $s' = s\{A^i \mapsto t\}$, où $(A,t) \in P$ et A^i est une instance de A dans s. Comme d'habitude, la fermeture réflexive et transitive de $\underset{G}{\longrightarrow}$ est notée $\underset{G}{\overset{*}{\longrightarrow}}$. Le langage engendré par G depuis l'axiome S est l'ensemble de mots de termes clos $L(G) = \{w \in T(F)^{|S|} \mid S \underset{G}{\overset{*}{\longrightarrow}} w\}$. Bien sûr, une grammaire dont l'axiome S est de taille n engendre uniquement des mots de termes de longueur n.

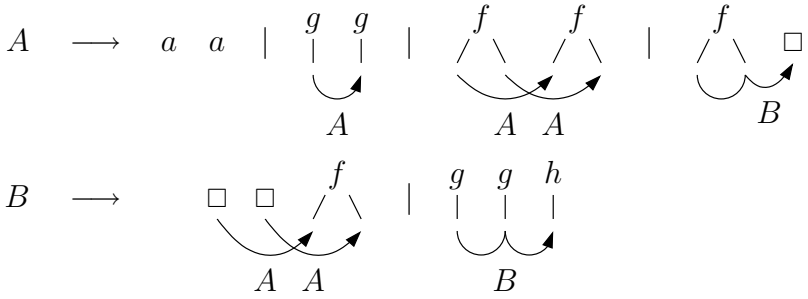
Dans [Rao97], on représente les grammaires par un certain type de grammaire à remplacement d'hyperarcs (cf. § 2.1.2.2). Soit (A,t) une règle de production, on voit A comme un hyperarc d'arité $|A|$ et t comme une forêt ordonnée dans laquelle toutes les variables appartenant à la même instance d'un non-terminal donné sont représentées par un hyperarc étiqueté par le non-terminal en question. Bien sûr, étant donné la définition des grammaires à multivariables, les grammaires de

graphes obtenues ont une forme très particulière, à laquelle elles doivent le nom de « forêts reliées par les feuilles ». Illustrons ceci sur un exemple.

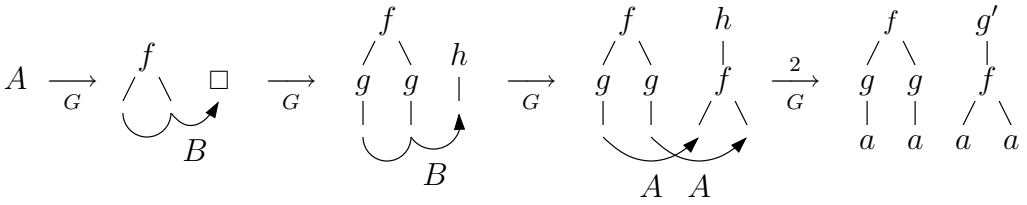
Exemple 3.8. Soient $A = A_1A_2$ et $B = B_1B_2B_3$ deux non-terminaux d'arités respectives 2 et 3. La grammaire G_1 de règles

$$\begin{aligned} A &\longrightarrow a\ a \mid gA_1\ gA_2 \mid fA_1^1A_1^2\ fA_2^1A_2^2 \mid fB_1B_2\ B_3 \\ B &\longrightarrow A_1^1\ A_1^2\ fA_2^1A_2^2 \mid gB_1\ gB_2\ hB_3 \end{aligned}$$

peut être représentée comme la grammaire à remplacement d'hyperarcs



dont une séquence de dérivation possible est :



Il s'agit juste d'une représentation visuelle des grammaires, et la définition de la dérivation ne change pas. On peut observer cependant que celle-ci est compatible avec la définition habituelle pour les grammaires à remplacement d'hyperarcs. Comme on peut s'y attendre, les grammaires à multivariables engendrent précisément les ensembles rationnels de mots de termes.

Théoreme 3.9 ([Rao97])**.** *Un ensemble de mots de termes est rationnel si et seulement si il est le langage engendré par une grammaire à multivariables.*

Exemple 3.10. L'ensemble de mots de termes engendré par G_1 depuis A peut être représenté par l'expression rationnelle

$$\big(\ (fxyz)\cdot_{xyz}(gx\ gy\ hz)^{*_{xyz}}(u^1\ u^2\ fv^1v^2) + (fu^1u^2\ fv^1v^2) + (gu\ gv) + (a\ a)\ \big)^{*_{uv}}.$$

3.1.4 Ensembles de mots de termes vus comme des relations binaires

Un ensemble rationnel de mots de termes de longueur n peut aussi être vu comme une relation binaire dans $T(F)^p \times T(F)^q$, avec $p + q = n$. Dans ce cas,

étant donné un non-terminal A, on définit la première et la seconde projection $\pi_1(A)$ et $\pi_2(A)$ de A par l'ensemble des variables appartenant à la première (resp. la seconde) projection de la relation. Une notation similaire est utilisée pour les parties droites de règles. Pour plus de clarté, on note une production (A,t) comme $(A,\pi_1(t) \times \pi_2(t))$. Sans perte de généralité, on supposera toujours que les non-terminaux sont définis de telle façon que $A = \pi_1(A)\pi_2(A)$ et $t = \pi_1(t)\pi_2(t)$. Par exemple, si l'axiome d'une relation dans $T(F)^p \times T(F)^q$ est $A = A_1 \dots A_n$, on peut supposer que $\pi_1(A) = A_1 \dots A_p$ et $\pi_2(A) = A_{p+1} \dots A_{p+q=n}$.

Exemple 3.11. La grammaire G_1 de l'exemple 3.8 engendre depuis le non-terminal A un langage $L(G_1,A) \in Rat_2$, qui peut être vu comme une relation de $T(F) \times T(F)$. Dans ce cas, on peut écrire ses règles

$$A \longrightarrow a \times a \mid gA_1 \times gA_2 \mid fA_1^1A_1^2 \times fA_2^1A_2^2 \mid fB_1B_2 \times B_3$$
$$B \longrightarrow A_1^1\ A_1^2 \times fA_2^1A_2^2 \mid gB_1\ gB_2 \times hB_3$$

Cette famille de relations est fermée par intersection avec des relations reconnaissables et par inverse mais pas par composition, et ne préserve pas la régularité en général, c'est à dire que l'image ou l'image inverse d'un langage régulier par une telle relation n'est pas nécessairement régulière. Cependant, [Rao97] isole une sous-famille de relations fermée par composition et préservant la régularité, qui sont appelées *transductions* rationnelles.

Définition 3.12 (Transduction rationnelle de termes)**.** Une relation rationnelle est une transduction si elle est engendrée par une grammaire pour laquelle il existe un entier k tel que, pour tout non-terminal A et toute dérivation $A \xrightarrow[G]{\geq k} t$, toutes les instances de multivariables ayant une occurrence dans t ont des variables dans au plus deux termes, un dans chaque projection de la relation.

3.2 Classification des systèmes de réécriture de termes

Dans le cas des mots, plusieurs classes naturelles de systèmes de réécriture peuvent être distinguées selon la façon dont leurs règles se chevauchent. Dans [Cau00], l'auteur considère la composition $\underset{R}{\longrightarrow} \circ \underset{R}{\longrightarrow}$ de deux étapes de réécriture, et énumère toutes les possibilités selon lesquelles la partie droite de la première règle appliquée et la partie gauche de la seconde peuvent se chevaucher. On dit que deux mots u et v se chevauchent si l'un est un facteur de l'autre, ou si un préfixe de l'un est un suffixe de l'autre. En interdisant les systèmes qui possèdent des chevauchements indésirables, on obtient quatre familles de systèmes dont il est ensuite démontré que la dérivation est rationnelle. Il s'agit des familles

de systèmes de réécriture de mots *gauches*, *droits*, *préfixes* et *suffixes*. De plus, il est montré que toute autre famille de systèmes vis-à-vis de ce critère de chevauchement des règles contient au moins un système dont la dérivation n'est pas rationnelle.

Comme les termes généralisent les mots, et que les relations non rationnelles de mots ne peuvent pas non plus être engendrés par des grammaires à multivariables, il nous suffit de nous concentrer sur l'extension de ces quatre familles de systèmes du cas des mots à celui des termes. On distinguera les familles de systèmes *ascendants* ou *top-down*, *descendants* ou *bottom-up*, *préfixes* et *suffixes*.

La notion de chevauchement que nous considérons est la même que pour les mots, à une légère nuance près en ce qui concerne les variables : deux termes se chevauchent si un préfixe de l'un est un sous-terme de l'autre, ou si l'un est un facteur de l'autre *à renommage des variables près*. Formellement, soit R un système de réécriture sur un alphabet F, un *chevauchement* entre une partie droite r et une partie gauche l de règles de R est un quintuplet (l,r,c,o,σ) tel que soit :

- $c[o] = r$, $o\sigma = l$ et o est non trivial (chevauchement descendant),
- $c[o] = l$, $o\sigma = r$ et o est non trivial (chevauchement ascendant),
- $o = l$, $c[o\sigma] = r$ et ni c ni σ ne sont triviaux (chevauchement domaine-interne strict),
- $o = r$, $c[o\sigma] = l$ et ni c ni σ ne sont triviaux (chevauchement image-interne strict).

Ces différents types de chevauchements incluent les 7 types de chevauchements initialement considérés dans le cas des mots (voir [Cau00] pour plus de détails). Par substitution non triviale, on entend ici spécifiquement une substitution dont l'application au terme considéré ajoute au moins un symbole fonctionnel (c'est à dire qu'on exclut en particulier les substitutions qui n'effectuent qu'un renommage des variables). On rappelle aussi que par hypothèse, on ne considère que des systèmes de réécriture dont l'ensemble de règles est clos par renommage des variables.

Étant donné un système R sur l'alphabet F, on considère l'ensemble O_R de tous les chevauchements entre parties droites et parties gauches de R. On notera qu'il peut exister plus d'un chevauchement possible pour une partie droite et une partie gauche données de R. Après avoir défaussé les systèmes où des chevauchements non souhaités se produisent, on isole les classes de systèmes suivantes. Un système de réécriture de termes R est dit :

- *descendant* si tout chevauchement de O_R est descendant (et uniquement descendant),
- *ascendant* si tout chevauchement de O_R est ascendant (et uniquement ascendant),

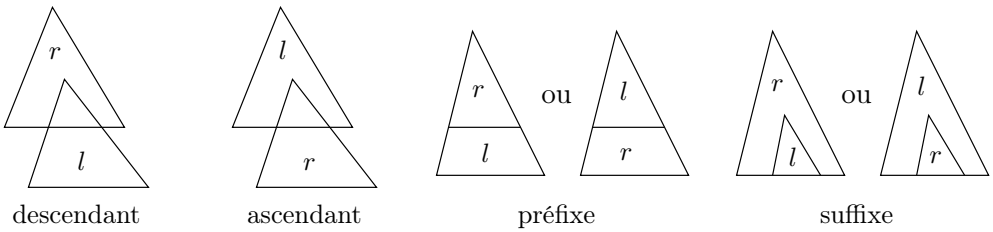


FIG. 3.1 – *Types de chevauchements dans les systèmes descendants, ascendants, préfixes et suffixes.*

– *préfixe* si tout chevauchement (l,r,c,o,σ) de O_R est tel que c est trivial,
– *suffixe* si tout chevauchement (l,r,c,o,σ) de O_R est tel que σ est trivial.

Notons que l'inverse d'un système descendant est ascendant (et réciproquement), que l'inverse d'un système préfixe est préfixe et que l'inverse d'un système suffixe est suffixe. La figure 3.1 illustre ces quatre types de systèmes. Les systèmes préfixes et suffixes généralisent respectivement les systèmes de réécriture par la racine et les systèmes clos.

Le prochain paragraphe établit la rationalité des dérivations des systèmes descendants (et donc aussi ascendants). Contrairement au cas des mots, où les systèmes préfixes et suffixes sont duaux et partagent les mêmes propriétés, la situation diffère dans le cas des termes. Le paragraphe 3.2.2 présente une courte démonstration du résultat bien connu que les systèmes préfixes sont aussi expressifs que des machines de Turing. Le paragraphe 3.2.3, cependant, montre que les systèmes suffixes ont une dérivation rationnelle.

3.2.1 Systèmes ascendants et descendants

Ce paragraphe traite de l'étude des systèmes de réécriture descendants et ascendants de termes et de leurs dérivations. Toutes les démonstrations que nous présentons s'intéressent aux systèmes descendants, mais les résultats s'étendent par inverse aux systèmes ascendants (cf. corollaire 3.18).

Pour tout système descendant linéaire et fini, nous montrons qu'il est possible de construire une grammaire engendrant sa relation de dérivation, ce qui implique que cette relation est rationnelle. De plus, d'après la forme de la grammaire, on observe que les dérivations de tels systèmes préservent la régularité des langages de termes. Des résultats similaires peuvent être obtenus pour les systèmes ascendants : la dérivation d'un système ascendant fini linéaire est rationnelle, et l'image inverse d'un langage régulier de termes par un tel système est également régulière.

3.2.1.1 Réécriture monotone

Observons premièrement que les systèmes descendants jouissent d'une certaine propriété de *monotonie*. Toute séquence de réécriture d'un tel système est équivalente à une séquence où chaque réécriture successive intervient à une position plus grande (au sens large) que la précédente. On appelle ceci la *réécriture descendante*. Soit R un système de réécriture de termes, on définit sa réécriture descendante

$$\xhookrightarrow[R]{*} = \bigcup_{n\geq 0} \xhookrightarrow[R]{n} \text{ avec } \begin{cases} \xhookrightarrow[R]{0} &= Id_{T(F)} \\ \xhookrightarrow[R]{n} &= \bigcup_{p_1,\ldots,p_n} \underset{u_1,v_1}{\longrightarrow}^{p_1} \circ \ldots \circ \underset{u_n,v_n}{\longrightarrow}^{p_n} \end{cases}$$

telle que les positions de réécriture ne décroissent pas en fonction des indices ($\forall i,j,\ i < j \ \Rightarrow\ \neg(p_j < p_i)$), et telle que si deux réécritures successives interviennent à des positions égales alors la seconde réécriture ne doit pas avoir une partie gauche triviale ($(p_i = p_{i-1}) \ \Rightarrow\ (u_i \notin X)$). Cette dernière condition signifie que, par exemple, la séquence

$$c[l\sigma] \underset{l,r}{\longrightarrow} c[r\sigma] \underset{x,r'}{\longrightarrow} c[r'\{x \mapsto r\sigma\}]$$

n'est pas descendante, parce que la seconde règle produit sa partie droite plus « haut » que la première. Les étapes de réécriture devraient être inversées pour obtenir la séquence descendante

$$c[l\sigma] \underset{x,r'}{\longrightarrow} c[r'\{x \mapsto l\sigma\}] \underset{l,r}{\longrightarrow}^{pos(x,c[r'])} c[r'\{x \mapsto r\sigma\}].$$

Le lemme suivant exprime le fait que, pour toute séquence de réécriture donnée d'un système descendant, les étapes de réécriture peuvent toujours être réordonnées en une séquence descendante équivalente.

Lemme 3.13 (Standardisation)**.** *Les relations de dérivation et de dérivation descendante d'un système descendant R coïncident :* $\underset{R}{\overset{*}{\longrightarrow}} = \xhookrightarrow[R]{*}$.

Démonstration. Par définition, $\xhookrightarrow[R]{*} \subseteq \underset{R}{\overset{*}{\longrightarrow}}$. Il reste à démontrer la réciproque, par la même technique que dans [Cau00]. Sans perte de généralité, on peut supposer que $R \cap (X \times X) = \emptyset$. Il est ensuite possible de réordonner toute dérivation en une dérivation descendante en appliquant une version modifiée de l'algorithme de tri à bulle, dans laquelle des suppressions ou ajouts d'éléments sont possibles. À

cette fin, nous utilisons les inclusions suivantes :

$$\underset{u,v}{\longrightarrow}^{\geqslant p} \circ \underset{x,v'}{\longrightarrow}^{p} \subseteq \underset{x,v'}{\longrightarrow}^{p} \circ \underset{u,v}{\longrightarrow}^{2^{>p}} \tag{3.1}$$

$$\underset{u,v}{\longrightarrow}^{>p} \circ \underset{u',v'}{\longrightarrow}^{p} \subseteq \underset{u',v'}{\longrightarrow}^{p} \circ \underset{u,v}{\longrightarrow}^{2^{\geqslant p}} \quad \text{avec } u,u' \notin X \tag{3.2}$$

$$\underset{u,v}{\longrightarrow}^{>p} \circ \underset{x,v'}{\longrightarrow}^{p} \subseteq \underset{x,v'}{\longrightarrow}^{p} \circ \underset{u,v}{\longrightarrow}^{2^{>p}} \quad \text{avec } u,v' \notin X \tag{3.3}$$

$$\underset{x,v}{\longrightarrow}^{>p} \circ \underset{u',y}{\longrightarrow}^{p} \subseteq \underset{u',y}{\longrightarrow}^{p} \circ \underset{x,v}{\longrightarrow}^{2^{>p}} \cup \underset{x,v}{\longrightarrow}^{p} \circ \underset{u',y}{\longrightarrow}^{2^{>p}} \tag{3.4}$$

où $\longrightarrow^{>p}$ (resp. $\longrightarrow^{\geqslant p}$) désigne une réécriture à toute position supérieure à (resp. supérieure ou égale à) p, et $\longrightarrow^{2^{>p}}$ ou $\longrightarrow^{2^{\geqslant p}}$ désigne une réécriture en plusieurs étapes à tout ensemble de telles positions. Démontrons maintenant la première inclusion. Soit

$$r \underset{u,v}{\longrightarrow}^{p} s \underset{x,v'}{\longrightarrow}^{q} t$$

avec $p \geqslant q$. On peut trouver des 1-contextes $\bar{r}$ et $\bar{s}$ et des substitutions ρ et σ tels que $r = \bar{r}[u\rho]$, $s = \bar{r}[v\rho] = \bar{s}[x\sigma]$ et $t = \bar{s}[v'\sigma]$, avec $pos(\Box,\bar{r}) = p$ et $pos(\Box,\bar{s}) = q$. Comme $p \geqslant q$ et par hypothèse sur R, il existe une substitution γ telle que $\bar{r} = \bar{s}[x\gamma]$ et $\sigma = \gamma[v\rho]$. Il est ainsi possible d'échanger les étapes de réécriture entre r et t :

$$r = (\bar{s}[x\gamma])[u\rho] \underset{x,v'}{\longrightarrow}^{q} (\bar{s}[v'\gamma])[u\rho] \underset{u,v}{\longrightarrow}^{P} (\bar{s}[v'\gamma])[v\rho] = t,$$

où $P \subseteq 2^{>q}$ est l'ensemble des positions auxquelles la variable $\Box$ apparaît dans $\bar{s}[v'\gamma]$, c'est à dire $P = pos(\Box,\bar{s}[v'\gamma])$. Donc

$$r \underset{x,v'}{\longrightarrow}^{q} \circ \underset{u,v}{\longrightarrow}^{2^{>q}} t,$$

et l'inclusion (3.1) est vérifiée. Plus généralement, considérons toute réécriture en deux étapes

$$r \underset{u,v}{\longrightarrow}^{p} s \underset{u',v'}{\longrightarrow}^{q} t$$

avec $p > q$. Par définition il doit exister deux 1-contextes $\bar{r}$ et $\bar{s}$ et deux substitutions ρ et σ tels que $r = \bar{r}[u\rho]$, $s = \bar{r}[v\rho] = \bar{s}[u'\sigma]$ et $t = \bar{s}[v'\sigma]$, avec $pos(\Box,\bar{r}) = \{p\}$ et $pos(\Box,\bar{s}) = \{q\}$. L'hypothèse que nous avons faite sur R implique certaines restrictions sur la structure de la configuration s.Comme R est descendant, $\bar{s}[u'] \leqslant \bar{r}$. Donc il doit exister une substitution γ telle que $\bar{r} = \bar{s}[u'\gamma]$ et $\sigma = \gamma[v\rho]$. Donc

$$r = (\bar{s}[u'\gamma])[u\rho] \underset{u',v'}{\longrightarrow}^{q} (\bar{s}[v'\gamma])[u\rho] \underset{u,v}{\longrightarrow}^{P} (\bar{s}[v'\gamma])[v\rho] = t,$$

où $P = pos(\Box,\bar{s}[v'\gamma])$. Ainsi

$$r \underset{u',v'}{\longrightarrow}^{q} \circ \underset{u,v}{\longrightarrow}^{P} t.$$

Comme $P \subseteq 2^{\geqslant q}$, l'inclusion (3.2) est vérifiée. Si $v' \notin X$ ou $v' = y \in X$ et $\gamma(y)$ est non-trivial, alors $P \subseteq 2^{>q}$, donc (3.3) et la première partie de (3.4) sont vraies. Enfin, si $u \in X$, $v' \in X$ et $\gamma(v')$ est trivial, alors $r = \bar{r}\rho$, $t = \bar{s}\sigma$, $\bar{r} = \bar{s}[u']$, $v\rho = \sigma$, donc

$$r = (\bar{s}[u'])\rho \underset{x,v}{\longrightarrow}^{q} (\bar{s}[v[u']])\rho \underset{u',x'}{\longrightarrow}^{P\subseteq 2^{>q}} (\bar{s}[v])\rho = t.$$

□

3.2.1.2 Rationalité des dérivations

Nous sommes maintenant prêts à démontrer la rationalité de la relation de dérivation de tout système de réécriture descendant. En utilisant la précédente propriété des systèmes descendants, il est possible de construire une grammaire à multivariables qui engendre directement la dérivation d'un tel système. Cette grammaire imite la façon dont un transducteur rationnel de mots fonctionne, en utilisant son état de contrôle pour conserver une mémoire finie des sous-termes lus ou encore à produire.

Théoreme 3.14. *Tout système de réécriture descendant fini linéaire R a une dérivation rationnelle.*

Démonstration. Soit R un système descendant fini linéaire. On note O l'ensemble de tous les chevauchements entre parties droites et parties gauches de R :

$$O = \{\ t \in C_n(F,X) \mid \exists\ s \in C_1(F,X), \\ u \in T(F,X)^n, s[t] \in Ran(R)\ \wedge\ t[u] \in Dom(R)\ \}. \quad (3.5)$$

Remarquons que le terme trivial $\Box$ appartient à O. Nous allons construire une grammaire G dont le langage engendré est exactement la relation de dérivation de R. Son ensemble fini de non-terminaux est $\{<*>\} \cup Q$, où $Q = \{\ <t> = <t>_1 \ldots <t>_{n+1} \mid t \in O\ \cap\ C_n(F)\}$ et pour tout $<t> \in Q$, $\pi_2(<t>)$ est une variable unique. Les règles de production de G sont de quatre types.
Type (1): $\forall\ f \in F_n$,

$$<\Box> \ \rightarrow\ f\ <\Box>_1^1 \ldots <\Box>_1^n\ \times\ f\ <\Box>_2^1 \ldots <\Box>_2^n$$
$$<*> \ \rightarrow\ f\ <*>^1 \ldots <*>^n$$

Type (2): $\forall\ t \in O \cap C_n(F)$, $t[u] \in O \cap C_m(F)$,

$$<t> \ \rightarrow\ u[\pi_1(<t[u]>)]\ \times\ \pi_2(<t[u]>)$$

Type (3): $\forall\ t[u] \in O$, $t \in O \cap C_\ell(F)$ (nécessairement $\{u_1, \dots u_\ell\} \subseteq O$),

$$<t[u]> \ \rightarrow\ \pi_1(<u_1>) \dots \pi_1(<u_\ell>) \times\ t[\pi_2(<u_1>) \dots \pi_2(<u_\ell>)]$$

Type (4): $\forall\ (t[u],s[v]) \in R$, $v = v_1 \dots v_\ell$ (nécessairement $\{t,v_1, \dots v_\ell\} \subseteq O$),

$$<t> \ \rightarrow\ u\sigma \ \times\ s[\pi_2(<v_1>) \dots \pi_2(<v_\ell>)]$$

où σ est un renommage de variables tel que pour toute variable x de u, $\sigma(x) = <v_i>_j$ si x est la j-ème variable apparaissant dans v_i (de gauche à droite), et $\sigma(x) = <*>$ si x n'apparaît dans aucun des v_i. La figure 3.2 illustre ces quatre types de règles.

Intuitivement, le rôle de cette substitution est de rassembler au sein du même non-terminal ou hyperarc toutes les variables de u qui appartiennent au même v_i, tout en respectant l'ordre dans lequel ces variables apparaissent dans v_i. De cette façon, on est assuré d'une instanciation correcte des non-terminaux de G. Si une variable de u n'apparaît pas dans v, ceci signifie qu'un sous-arbre entier est défaussé par la règle de réécriture en cours d'application. Donc la règle de la grammaire doit accepter un sous-arbre quelconque à cette position, ce qui est assuré par le non-terminal unaire $<*>$.

Par simplicité, on considère uniquement des règles de type (4) dans lesquelles $t,v_1, \dots v_\ell$ sont *maximaux*. Les autres cas peuvent être simulés par des compositions finies adaptées de règles de type (2), (3) et (4). □

Remarque 3.15. Il devrait être possible d'étendre cette démonstration sans trop de difficulté au cas des systèmes descendants possédant un ensemble reconnaissable de règles : au lieu d'être fini, l'ensemble O des chevauchements entre parties droites et gauches est un langage régulier de termes. Il serait donc suffisant, au lieu d'associer un élément de O à chaque non-terminal de G, d'associer à ces non-terminaux un état de contrôle d'un automate descendant reconnaissant O.

Exemple 3.16. Considérons le système descendant linéaire R sur l'alphabet $F = \{f^{(2)}, g^{(1)},h^{(1)},a^{(0)}\}$ muni de l'unique règle $fgxgy \rightarrow hfxy$. La grammaire correspondante est celle de l'exemple 3.11 où chaque non-terminal représente l'un des chevauchements possibles des règles de R : A représente $\Box$ et B représente $f\Box_1\Box_2$. Notons que les règles de type (4) avec des chevauchements non maximaux ne sont pas considérés. Cet exemple illustre aussi le fait que l'image inverse d'un langage régulier de termes par la dérivation d'un système descendant n'est pas nécessairement régulier. Par exemple, l'image par R_G^{-1} de h^*faa est le langage $\{h^*fg^nag^na \mid n \geq 0\}$, qui n'est pas régulier.

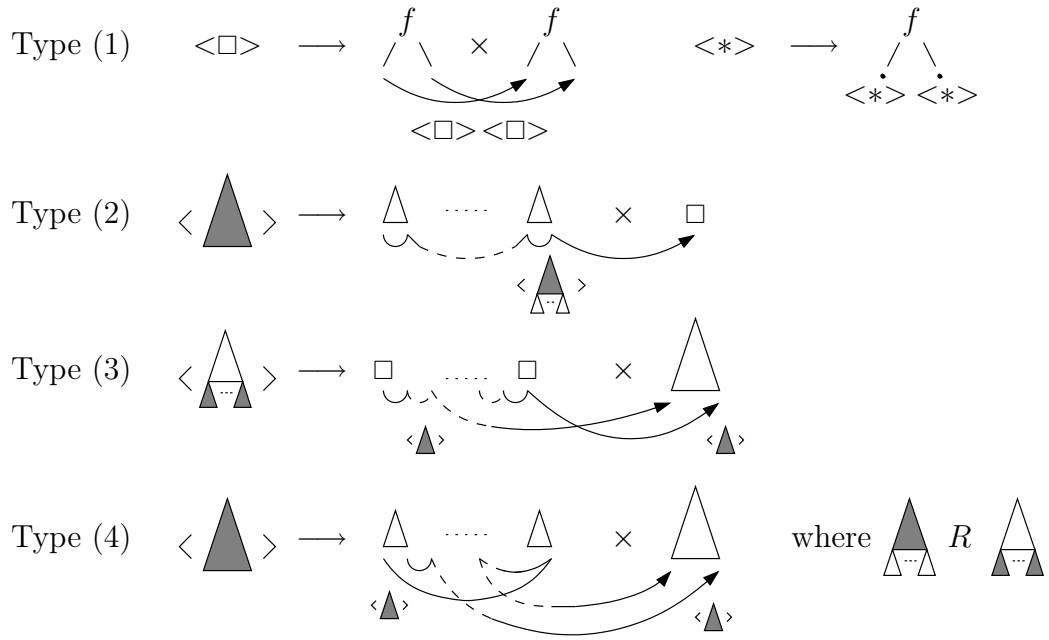


FIG. 3.2 – *Grammaire associée à un système descendant.*

3.2.1.3 Préservation de la régularité

Nous mentionnons ici une propriété des systèmes descendants déjà connue depuis quelques années pour des classes plus grandes de systèmes, mais que nous pouvons néanmoins observer très simplement dans le cas présent, et dont nous fournissons une construction directe.

Proposition 3.17. *L'image de tout langage régulier de termes par la dérivation d'un système de réécriture descendant fini linéaire de termes est régulière.*

Démonstration. Remarquons tout d'abord que le domaine de la dérivation d'un système descendant est $T(F)$, et que son co-domaine est régulier : la grammaire obtenue dans la démonstration du théorème 3.14 n'a que des non-terminaux dont la seconde projection est réduite à une variable.

Soit L un langage régulier accepté par un certain automate descendant d'arbres A avec un ensemble Q d'états de contrôle. Soit R un système descendant fini linéaire. D'après la démonstration précédente, on peut construire une grammaire G engendrant $\xrightarrow[R]{*}$. Avant de poursuivre, définissons une grammaire « produit » G_A de domaine L et dont le co-domaine est $\xrightarrow[R]{*}(L)$. Les non-terminaux de G_A seront de la forme

$$(N_1,q_1)\dots(N_n,q_n)(N_{n+1},q_1\dots q_n), \tag{3.6}$$

et notés simplement $N_{q_1 \dots q_n}$. Pour chaque production

$$N_1 \dots N_{n+1} \longrightarrow t_1 \dots t_n \; s$$

de G, la grammaire produit G_A est munie de toutes les productions

$$N_{q_1 \dots q_n} \longrightarrow t'_1 \dots t'_n \; s'$$

où le mot de termes m-contextuel $t_1 \dots t_n$ est partiellement accepté par A (cf. § 1.3.5.2) depuis le mot de contrôle initial $(q_1 \dots q_n)$ jusqu'au mot de contrôle final $(q'_1 \dots q'_m)$. De plus, $t'_1 \dots t'_n$ est obtenu depuis $t_1 \dots t_n$ en associant par paires chacune de ses m variables avec le composant associé du mot de contrôle final, et s' est obtenu depuis s en associant par paires chacune de ses variables avec un mot de contrôle de Q^*, de façon à compléter chaque instance des non-terminaux de G_A conformément à (3.6). Il résulte de cette opération qu'un couple (s,t) appartient à $L(G_A)$ si et seulement si $(s,t) \in L(G)$ et $s \in L$. En d'autres termes, la seconde projection de G_A est exactement l'ensemble des termes qui sont l'image par $\xrightarrow[R]{*}$ d'un certain terme de L, et par conséquent $\pi_2(L(G_A)) = \xrightarrow[R]{*}(L)$. En oubliant la projection gauche de chaque production de la grammaire, on obtient une grammaire dont chaque non-terminal a une arité de 1. De telles grammaires sont appelées *grammaires régulières de termes* et engendrent des langages réguliers de termes. Dans le cas présent, on obtient une grammaire régulière engendrant $\xrightarrow[R]{*}(L)$. □

L'inverse d'un système descendant est, par définition, ascendant. Pour tout système descendant R on peut construire une grammaire G engendrant $\xrightarrow[R]{*}$. Par conséquent, la grammaire $\pi_2(G)\pi_1(G)$ obtenue en inversant les deux projections de G engendre la dérivation $\xrightarrow[R^{-1}]{*}$ du système ascendant R^{-1}. La préservation par inverse de la régularité s'ensuit.

Corollaire 3.18. *Tout système de réécriture ascendant fini linéaire de termes R a une dérivation $\xrightarrow[R]{*}$ rationnelle, et l'image inverse par $\xrightarrow[R]{*}$ de tout langage régulier de termes est régulière.*

La question de trouver des classes de systèmes de réécriture de termes qui préservent la régularité des langages de termes a été traitée par plusieurs auteurs. Les systèmes descendants forment une sous-famille stricte des *systèmes semi-monadiques généralisés* [GV98], qui sont eux-mêmes strictement inclus dans celle des *systèmes linéaires à droite à chevauchements de chemins finis* [TKS00]. Il a été montré de ces deux classes qu'elles préservent la régularité. Par conséquent, c'est aussi le cas des systèmes descendants. Cependant, il devrait être dit qu'aucune de ces deux classes n'a une dérivation rationnelle. Par exemple, le système

semi-monadique généralisé d'unique règle $gx \longrightarrow fgfx$ n'a clairement pas une dérivation rationnelle : son intersection avec la relation reconnaissable $ga \times f^*gf^*a$ est $ga \times \{f^ngf^na | n \geq 0\}$. Par des arguments classiques de pompage (adaptés à ce nouveau cadre), cette relation n'est pas rationnelle.

3.2.2 Systèmes préfixes

Les systèmes de réécriture par la racine sont déjà connus comme étant très expressifs : en effet, ils peuvent simuler les étapes d'exécution de machines de Turing. Ceci implique un résultat négatif direct concernant leurs relations de dérivation.

Proposition 3.19. *Il existe un système de réécriture préfixe linéaire et fini de termes dont la dérivation est non récursive.*

Démonstration. Soit $M = (\Sigma,\Gamma, [,],Q,q_0,F,\delta)$ une machine de Turing possédant uniquement des règles étiquetées par ε, construisons un système préfixe R_M sur l'alphabet $Q \cup \Gamma \cup \{[,],\bot\}$, à variables dans $\{x,y\}$, où les symboles de Q sont considérés comme binaires, les symboles de $\Gamma \cup \{[,]\}$ comme unaires et où $\bot \notin \Gamma$ est une constante. On traduit les règles de M en règles de réécriture de R_M de la façon suivante :

$$\begin{aligned}
\forall pA \xrightarrow{\varepsilon} qB+ \in \delta, &\qquad (pxAy,qBxy) \in R_M,\\
\forall pA \xrightarrow{\varepsilon} qB \in \delta, &\qquad (pxAy,qxBy) \in R_M,\\
\forall pA \xrightarrow{\varepsilon} qB- \in \delta,\ C \in (\Gamma \cup \{[\,\}), &\qquad (pCxAy,qBCxy) \in R_M,\\
\forall pA \xrightarrow{\varepsilon} qBA \in \delta, &\qquad (pxAy,qxBAy) \in R_M,\\
\forall pA \xrightarrow{\varepsilon} q \in \delta, &\qquad (pxAy,qxy) \in R_M.
\end{aligned}$$

Ce système a à la fois des chevauchements de la forme $l = r\sigma$ et $r = l\sigma$, pour des parties gauche et droite l et r et une substitution σ quelconques. Il s'agit donc d'un système préfixe, et qui n'est ni descendant, ni ascendant, ni suffixe. Il est clair que calculer la relation de dérivation de R_M est au moins aussi difficile que de calculer la relation d'accessibilité dans M, et est donc indécidable. Par conséquent $\xrightarrow[R]{*}$ est non récursive, et donc évidemment pas non plus rationnelle. □

3.2.3 Systèmes suffixes

La famille des systèmes de réécriture clos de termes, qui est une sous-famille des systèmes suffixes, a déjà été étudiée par plusieurs auteurs. En particulier, Dauchet et Tison [DT85] ont montré que les dérivations des systèmes clos peuvent être reconnues par un certain type d'automates à composantes appelés *transducteurs*

d'arbres clos (GTT). Nous utilisons des arguments similaires afin de démontrer que, d'une façon plus générale, tout système suffixe a une dérivation rationnelle. Après avoir introduit une propriété des systèmes suffixes vis-à-vis de la notion de *réécriture suffixe*, nous montrons que la relation de dérivation de tout système suffixe est rationnelle. Enfin, nous établissons la preuve que l'image et l'image inverse de tout langage régulier de termes par la dérivation d'un système suffixe sont régulières, et qu'il est possible de construire de façon effective un automate les reconnaissant.

3.2.3.1 Réécriture suffixe

Les systèmes de réécriture clos sont les systèmes dont les règles ne contiennent pas de variables. Par conséquent, ils peuvent seulement réécrire des sous-termes (suffixes) du terme fourni. La réécriture suffixe peut être vue comme l'application d'une sorte de règles d'automate à pile sur une pile arborescente, où les seules portions autorisées à la modification sont les parties environnant les feuilles (ou constantes).

Étant donné un système de réécriture non clos R sur F avec des variables dans X, on définit une notion restreinte de réécriture par R sur les termes de $T(F \cup Y)$, où Y est un ensemble quelconque de variables. Cette notion, inspirée par la réécriture close et appelée réécriture suffixe, considère temporairement les variables de Y comme autant de constantes, tandis que la substitution des variables des règles de R est restreinte aux éléments de Y. Nous appelons ceci réécriture suffixe, puisque une telle restriction n'autorise l'application des règles de réécriture que sur des suffixes des termes considérés.

Définition 3.20. Pour tout système de réécriture R sur F à variables dans X, et pour tout ensemble de variables Y disjoint de F_0, on définit la relation de *réécriture suffixe* de R comme

$$\xrightarrow[R]{} \;=\; \left\{ (c[l\sigma],c[r\sigma]) \in T(F \cup X)^2 \mid (l,r) \in R \wedge\ c \in C_1(F) \wedge\ \sigma \in Y^X \right\}$$

où Y^X est l'ensemble des applications (ou substitutions) de X vers Y.

La raison pour laquelle on ne traite pas simplement Y comme un ensemble de nouvelles constantes, ce qui reviendrait à associer à tout système de réécriture un système clos sur $F \cup Y$, est que nous souhaitons comme d'habitude que deux termes ne différant que par un nommage de leurs variables (dans Y) soient considérés égaux.

3.2.3.2 Décomposition des dérivations

Les systèmes suffixes ont un comportement particulier par rapport à leur réécriture suffixe. En effet, il est toujours possible de réordonner la dérivation de

tout terme t en un terme s par un système suffixe afin de pouvoir la décomposer en deux phases. Premièrement, un préfixe $\bar{t}$ de t est lu, et un certain nombre d'étapes de réécriture suffixe y sont appliquées. Une fois que cette première séquence de réécritures est terminée, $\bar{t}$ a été réécrit en un préfixe $\bar{s}$ de s, pour ne jamais plus être modifié au cours de la séquence de dérivation. Dans un second temps, le reste de t est dérivé selon la même décomposition en deux phases, jusqu'à ce que l'ensemble des réécritures ait été effectué. Par conséquent, on peut dire que la dérivation d'un système suffixe est équivalente à sa dérivation suffixe « itérée ».

Lemme 3.21. *Pour tout système de réécriture suffixe de termes R,*

$$s \xrightarrow[R]{k>0} t \iff \begin{cases} \exists\ \bar{s},\bar{t} \in T(F,X),\ \sigma,\tau \in S(F,X),\ k' > 0,\ s = \bar{s}\sigma\ \wedge\ t = \bar{t}\tau \\ \wedge\ \bar{s} \xrightarrow[R]{k'}\!\!\!\!\rightarrow \bar{t}\ \wedge\ \forall x \in Var(\bar{s}) \cap Var(\bar{t}),\ \sigma(x) \xrightarrow[R]{\leq k-k'} \tau(x). \end{cases}$$

Démonstration. Remarquons tout d'abord que comme $s \xrightarrow[R]{}\!\!\!\!\rightarrow t \implies s\sigma \xrightarrow[R]{} t\sigma$ pour toute substitution close σ, l'implication réciproque est triviale. Démontrons l'implication directe par récurrence sur k :

$k = 1$: $s \xrightarrow[R]{} t$ implique qu'il existe un contexte c et une substitution σ tels que $s = c[l\sigma]$ et $t = c[r\sigma]$. Donc par définition de la réécriture suffixe $c[l] \xrightarrow[R]{}\!\!\!\!\rightarrow c[r]$, et pour toute variable x on a bien sûr $\sigma(x) \xrightarrow[R]{0} \sigma(x)$.

$k \Rightarrow k+1$: soit $s \xrightarrow[R]{k} s' \xrightarrow[l,r]{}\!\!\!\!\rightarrow^{p} t$ avec lRr, $s = \bar{s}\sigma$ et $s' = \bar{s'}\sigma'$. On distingue deux cas :

$p \in Pos(\bar{s'})$: s'il existe un contexte c tel que $\bar{s'} = c[l]$, alors on a $\bar{s} \xrightarrow[R]{k}\!\!\!\!\rightarrow \bar{s'} \xrightarrow[R]{}\!\!\!\!\rightarrow c[r]$ et la condition est vérifiée pour $k' = k$ et $t = (c[r])\sigma'$. Si ce n'est pas le cas, alors il doit exister un contexte c et une substitution non-triviale ω' telle que $\bar{s'} = (c[l])\omega'$. Comme R est suffixe et comme, par hypothèse de récurrence, $\bar{s} \xrightarrow[R]{*}\!\!\!\!\rightarrow \bar{s'}$, il doit exister ω telle que $\bar{s} = (c[l])\omega$ et pour toute variable x commune à l et r, $\omega(x) \xrightarrow[R]{*}\!\!\!\!\rightarrow \omega'(x)$. On peut donc écrire s sous la forme $(c[l])\omega\sigma$, t sous la forme $(c[r])\omega'\sigma'$, et vérifier que la condition est vérifiée avec $k' = 1$.

$p \notin Pos(\bar{s'})$: par hypothèse de récurrence, on peut trouver $k' > 0$ tel que pour tout x commun à $\bar{s}$ et $\bar{s'}$, $\sigma(x) \xrightarrow[R]{\leq k-k'} \sigma'(x)$. De plus, en appliquant la règle (l,r) à l'un des termes $\sigma'(x)$, on obtient $\sigma'(x) \xrightarrow[l,r]{} \tau(x)$. On a donc $t = \bar{s'}\tau$, $\bar{s} \xrightarrow[R]{k'}\!\!\!\!\rightarrow \bar{s'}$ et $\sigma(x) \xrightarrow[R]{k-k'+1} \tau(x)$ pour tout x commun à $\bar{s}$ et $\bar{s'}$, ce qui vérifie la condition et clôt la démonstration.

□

Une autre propriété intéressante est que, pour tout système reconnaissable linéaire, une séquence de réécriture suffixe est toujours équivalente à une séquence en deux parties, dont la première ne fait que consommer des sous-termes suffixes du terme d'entrée, et dont la seconde ne fait que produire de nouveaux sous-termes à leur place.

Lemme 3.22. *Pour tout système de réécriture reconnaissable et linéaire R de termes sur F et X, il existe un alphabet gradué fini Q et trois systèmes de réécriture finis*

- $R_- \subseteq \{px \longrightarrow fp_1x_1 \dots p_nx_n \mid f \in F,\ p,p_1,\dots,p_n \in Q,\ x,x_1,\dots,x_n \in X^*\}$
- $R_= \subseteq \{px \longrightarrow qy \mid p,q \in Q,\ x,y \in X^*\}$
- $R_+ \subseteq \{fp_1x_1 \dots p_nx_n \longrightarrow px \mid f \in F,\ p,p_1,\dots,p_n \in Q,\ x,x_1,\dots,x_n \in X^*\}$

tels que $s \xrightarrow[R]{*} t \iff s \xrightarrow[R_+ \cup R_=]{*} \circ \xrightarrow[R_- \cup R_=]{*} t$.

Démonstration. Soit R un système de réécriture reconnaissable linéaire sur F et X. Soit $(A_i,B_i)_{i\in[1,l]}$ un ensemble de couples d'automates descendants, chacun acceptant un ensemble de règles de R. L'ensemble des états de contrôle de A_i (resp. B_i) sera noté P_i (resp. Q_i). Sans perdre de généralité, nous supposons que tous les P_i et Q_i sont disjoints entre eux. Nous définissons aussi l'union P de tous les ensembles P_i et l'union Q de tous les Q_i. Pour tout état $p \in P \cup Q$, soit $\nu(p)$ l'ensemble de toutes les frontières de variables possibles dans le langage accepté depuis l'état initial p par l'automate correspondant, c'est à dire l'ensemble de tous les n-uplets de variables apparaissant dans les termes de ce langage, lus de la gauche vers la droite. Comme le nombre total de variables du système est fini et que les systèmes considérés sont linéaires, on peut supposer que $|\nu(p)| = 1$ pour tout p (si ce n'est pas le cas, il suffit d'ajouter un état de contrôle à l'automate pour chacune des frontières possibles, qui sont en nombre fini).

Dans un premier temps, nous définissons un nouveau système de réécriture R' sur $F \cup P \cup Q$ et X. Les alphabets d'états P et Q sont considérés comme des alphabets gradués dont les symboles sont tous d'arité égale au nombre de variables apparaissant dans les termes du langage associé (qui est, nous le rappelons, supposé unique). Par exemple, supposons que depuis l'état p l'automate A accepte le langage g^*fxy. Dans ce cas, p sera considéré comme un symbole binaire (et $\nu(p) = xy$).

Nous donnons à R' l'ensemble de règles suivant. Pour toute transition $pf \to p_1 \dots p_m$ dans un certain A_i telle que $\nu(p) = \nu(p_1) \dots \nu(p_m) = x_1 \dots x_n$, on a :

$$fp_1\nu(p_1) \dots p_m\nu(p_m) \ \to\ px_1 \dots x_n \ \in\ R'.$$

Les règles de cette forme permettent à R' de consommer une partie gauche de règle de R dans le terme d'entrée. Pour toute transition $qf \to q_1 \dots q_m$ d'un

certain B_i telle que $\nu(q) = \nu(q_1) \dots \nu(q_m) = x_1 \dots x_n$, on a:

$$qx_1 \dots x_n \ \rightarrow \ fq_1\nu(q_1) \dots q_m\nu(q_m) \ \in \ R'.$$

Les règles de cette forme permettent à R' de produire une partie droite de règle de R dont une partie gauche correspondante a été précédemment consommée. Remarquons que, dans ces deux derniers cas, si $f = x$ pour une certaine variable x on obtient des règles de la forme $x \rightarrow px$ et $qx \rightarrow x$ respectivement. Enfin, pour tout couple (p_0,q_0) d'états initiaux d'un certain couple (A_i,B_i), avec $\nu(p_0) = x_1 \dots x_n$ et $\nu(q_0) = x_{k_1} \dots x_{k_m}$ on a :

$$p_0x_1 \dots x_n \ \rightarrow \ q_0x_{k_1} \dots x_{k_m} \ \in \ R'.$$

Ceci simule l'application d'une règle de réécriture de l'ensemble $L(A_i) \times L(B_i)$ en initiant un calcul de l'automate B_i quand un calcul acceptant inversé de A_i a été effectué avec succès. Une fois restreinte au domaine $T(F)^2$, la relation de dérivation de R' coïncide avec celle de R :

$$\forall s,t \in T(F),\ s \ \xrightarrow[R]{*} \ t \iff s \ \xrightarrow[R']{*} \ t. \tag{3.7}$$

La démonstration de cette propriété ne présente aucune difficulté particulière et ne sera donc pas détaillée. Dans le reste de la démonstration, p,q et toutes leurs variations désigneront des états de contrôle d'automates dans $P \cup Q$. Les mots de variables dans X^* seront notées u,v,u_i,v_i, …, et σ désignera un renommage de variable.

Dans un second temps, nous définissons R_+ comme $\{fp_1u_1 \dots p_nu_n \, R' \, pu\}$ (règles de « consommation »), R_- comme $\{qv \, R' \, fq_1v_1 \dots q_nv_n\}$ (règles de « production ») et $R_=$ comme la plus petite relation binaire dans $T(F,X)^2$ fermée par les règles d'inférence suivantes :

$$\frac{}{pu \ R_= \ pu} \quad (1) \qquad \frac{pu \ R' \ qv}{pu \ R_= \ qv} \quad (2)$$

$$\frac{pu \ R_= \ qv}{pu\sigma \ R_= \ qv\sigma} \quad (3) \qquad \frac{qu \ R_= \ q'v \quad q'v \ R_= \ q''z}{qu \ R_= \ q''z} \quad (4)$$

$$\frac{pu \ R_- \ fp_1u_1 \dots p_nu_n \quad fq_1v_1 \dots q_nv_n \ R_+ \ qv \quad \forall i, p_iu_i \ R_= \ q_iv_i}{pu \ R_= \ qv} \quad (5)$$

Comme F,P,Q et X sont finis, et comme chaque symbole p de $P \cup Q$ a une arité bien définie, $R_=$ est fini et calculable de façon effective. Mentionnons une propriété simple de $R_=$:

$$\forall\, pu,t,qv \in T(F,X),\ pu \ \xrightarrow[R_- \cup R_=]{*} \ t \ \xrightarrow[R_+ \cup R_=]{*} \ qv \implies pu \, R_= \, qv. \tag{3.8}$$

Ceci peut être prouvé par induction sur la hauteur k du terme t :

$k = 0$: aucune règle de R_- ou R_+ n'est appliquée, donc $pu \xrightarrow[R_=]{*} qv$. Ainsi, d'après la règle d'inférence (4), la propriété est vraie.

$k \Rightarrow k+1$: décomposons la séquence de dérivation entre s et t :

$$pu \xrightarrow[R_=]{} p'u' \xrightarrow[R_-]{} fp_1u_1 \dots p_nu_n$$
$$\xrightarrow[R_-\cup R_=]{*} \circ \xrightarrow[R_+\cup R_=]{*} fq_1v_1 \dots q_nv_n \xrightarrow[R_+]{} q'v' \xrightarrow[R_=]{} qv.$$

Par hypothèse de récurrence, $fp_1u_1 \dots p_nu_n \xrightarrow[R_=]{*} fq_1v_1 \dots q_nv_n$, donc d'après la règle d'inférence (5), $p'u' \, R_= \, q'v'$. Enfin, d'après la règle d'inférence (4), $pu \, R_= \, qv$.

Il reste à démontrer que R_+, $R_=$ et R_- vérifient bien la propriété énoncée. D'après (3.7), il suffit de montrer que $\xrightarrow[R_+\cup R_=]{*} \circ \xrightarrow[R_-\cup R_=]{*} = \xrightarrow[R']{*}$. L'inclusion directe équivaut à $\xrightarrow{R_=} \subseteq \xrightarrow[R']{*}$. Supposons que $s \xrightarrow{R_=} t$ pour deux termes s et t quelconques. La règle de $R_=$ utilisée dans cette étape de réécriture peut seulement être définie à l'aide des règles d'inférence ci-dessus. On peut donc raisonner par récurrence sur la séquence de règles d'inférence utilisée pour la définir pour démontrer qu'il doit s'ensuivre que $s \xrightarrow[R']{*} t$. Ce raisonnement ne pose pas de difficulté et n'est pas détaillé ici.

Pour montrer la réciproque nous allons établir, par récurrence sur k, la propriété suivante :

$$\forall k,\ s \xrightarrow[R']{k} t \implies \exists\, c \in C_1(F),\ (q_iu_i)_{i\in[n]} \in T(F,X),$$
$$s \xrightarrow[R_+\cup R_=]{*} c[q_1u_1 \dots q_nu_n] \xrightarrow[R_-\cup R_=]{*} t.$$

$k = 0$: $s = t = c$, $n = 0$, donc trivialement $s \xrightarrow[R_+\cup R_=]{*} c \xrightarrow[R_-\cup R_=]{*} t$.

$k \Rightarrow k+1$: soit $s \xrightarrow[R']{k} t \xrightarrow[l,r]{} t'$ avec $lR'r$. Par hypothèse de récurrence,

$$s \xrightarrow[R_+\cup R_=]{*} c[q_1u_1 \dots q_nu_n] \xrightarrow[R_-\cup R_=]{*} t = c[t_1 \dots t_n].$$

Trois cas sont possibles :

– $\exists\, c'$, $t = c'[t_1 \dots t_n\, l]$: dans ce cas s peut s'écrire $c'[s_1 \dots s_n\, l]$, donc la

séquence de dérivation suivante est valide :

$$\begin{aligned} s &\xrightarrow[R_+\cup R_=]{*} c'[q_1u_1\dots q_nu_n\,l] \xrightarrow[R_+]{} c'[q_1u_1\dots q_nu_n\,q_lu] \\ &\xrightarrow[R_=]{} c'[q_1u_1\dots q_nu_n\,q_rv] \ (\text{car } l\,R'\,r) \\ &\xrightarrow[R_-]{} c'[q_1u_1\dots q_nu_n\,r] \xrightarrow[R_-\cup R_=]{*} c'[t_1\dots t_n\,r] = t'. \end{aligned}$$

– $\exists\, i,\ t_i = \bar{t}_i[l]$: la seule façon de produire t_i depuis q_iu_i est selon les étapes

$$q_iu_i \xrightarrow[R_-\cup R_=]{*} \bar{t}_i[q_lu] \xrightarrow[R_-]{} t_i.$$

Donc la séquence de dérivation suivante est valide :

$$\begin{aligned} s &\xrightarrow[R_+\cup R_=]{*} c[q_1u_1\dots q_nu_n] \xrightarrow[R_-\cup R_=]{*} c[q_1u_1\dots \bar{t}_i[q_lu]\dots q_nu_n] \\ &\xrightarrow[R_=]{} c[q_1u_1\dots \bar{t}_i[q_rv]\dots q_nu_n] \xrightarrow[R_-]{} t'. \end{aligned}$$

– $\exists\, c', j > i \geq 1,\ t = c'[t_1\dots t_i\,l\,t_{j+1}\dots t_n]$, avec $l = \bar{l}[t_{i+1}\dots t_j]$. La séquence de dérivation entre s et t' peut alors s'écrire

$$\begin{aligned} s &\xrightarrow[R_+\cup R_=]{*} c[q_1u_1\dots q_nu_n] \xrightarrow[R_-\cup R_=]{*} c[t_1\dots t_n] \\ &\xrightarrow[R_+]{*} c[t_1\dots t_i\,q'_{i+i}u'_{i+1}\dots q'_ju'_j\,t_{j+1}\dots t_n] \\ &\xrightarrow[R_+]{} c'[t_1\dots t_i\,q_lu\,t_{j+1}\dots t_n] \xrightarrow[R_=]{} c'[t_1\dots t_i\,q_rv\,t_{j+1}\dots t_n] \\ &\xrightarrow[R_-]{} c'[t_1\dots t_i\,r\,t_{j+1}\dots t_n] = t' \end{aligned}$$

Remarquons que pour tout $k \in [i+1,j]$, $q_ku_k \xrightarrow[R_-\cup R_=]{*} t_k \xrightarrow[R_+\cup R_=]{*} q'_ku'_k$. Donc d'après (3.8), $q_ku_k\,R_=\,q'_ku'_k$, et donc

$$\begin{aligned} s &\xrightarrow[R_+\cup R_=]{*} c[q_1u_1\dots q_nu_n] \xrightarrow[R_+\cup R_=]{*} c'[q_1u_1\dots t_i\,q_lu\,q_{j+1}u_{j+1}\dots q_nu_n] \\ &\xrightarrow[R_-\cup R_=]{*} c'[t_1\dots t_i\,r\,t_{j+1}\dots t_n] = t'. \end{aligned}$$

Ceci conclut la démonstration du lemme 3.22. □

Le lemme 3.22 peut être reformulé de la façon suivante : un couple de termes (s,t) appartient à la dérivation du système R si et seulement si il existe un contexte c tel que $s = c[s_1\dots s_n]$, $t = c[t_1\dots t_n]$ et pour tout $i \in [1,n]$, il existe un terme q_ix_i tel que $s_i \xrightarrow[R_+\cup R_=]{*} q_ix_i$ et $q_ix_i \xrightarrow[R_-\cup R_=]{*} t_i$.

3.2.3.3 Rationalité de la dérivation

Maintenant que la structure des dérivations de systèmes suffixes est mieux comprise, nous sommes en mesure de construire une grammaire engendrant la relation de dérivation d'un système suffixe quelconque.

Théoreme 3.23. *Tout système de réécriture suffixe linéaire reconnaissable de termes R a une dérivation rationnelle.*

Démonstration. Soit R un système suffixe linéaire reconnaissable sur $T(F,X)$. Soient R_+, $R_=$ et R_- les systèmes de réécriture construits au lemme 3.22. Soit N un ensemble de paires de la forme $u|v$ où u et v sont deux mots de termes linéaires dans $Ran(R_+ \cup R_=)^*$ et $Dom(R_- \cup R_=)^*$ respectivement. Rappelons que Ran et Dom sont définis à renommage des variables près. On peut donc imposer à u et v de partager le même ensemble de variables ($Var(u) = Var(v)$), et qu'il n'existe aucun couple de sous-mots stricts u' et v' de u et v tels que $Var(u') \neq Var(v')$ (c'est à dire qu'il ne doit pas être possible de séparer $u|v$ en deux non-terminaux corrects). Ajouté au fait que F est fini et u et v sont linéaires, ceci implique que N est fini pour un certain renommage des variables fixé. Donc, étant donné un axiome I, on peut construire une grammaire G dont l'ensemble de non-terminaux est $N \cup \{I,I'\}$, possédant l'ensemble fini de productions suivant :

$$\forall\ f \in F, \qquad I \longrightarrow f I_1^1 \dots I_1^n \times f I_2^1 \dots I_2^n \quad \text{et} \quad I' \longrightarrow f I'^1 \dots I'^n \tag{3.9}$$

$$\forall\ px \in Dom(R_- \cup R_=) \cap Ran(R_+ \cup R_=), \qquad I \longrightarrow px|px \tag{3.10}$$

$$\forall\ u' \xrightarrow[R_=]{*} u, \quad v \xrightarrow[R_=]{*} v', \quad u' \in Ran(R_+)^*, \quad v' \in Dom(R_-)^*, \qquad u|v \longrightarrow u'|v' \tag{3.11}$$

$$\forall\ u_1 = p_1 x_1 \dots p_i x_i, \quad u_2 = p_{j+1} x_{j+1} \dots p_n x_n, \quad v = q_1 y_1 \dots q_m y_m, \quad f p_{i+1} x_{i+1} \dots p_j x_j \, R_+ \, px, \qquad u_1\, px\, u_2|v \longrightarrow \mu_1 \dots \mu_i \, (f \mu_{i+1} \dots \mu_j) \, \mu_{j+1} \dots \mu_n \times \nu_1 \dots \nu_m \tag{3.12}$$

$$\forall\ u = p_1 x_1 \dots p_n x_n, \quad v_1 = q_1 y_1 \dots q_i y_i, \quad v_2 = q_{j+1} y_{j+1} \dots q_m y_m, \quad qy \, R_- \, f q_{i+1} y_{i+1} \dots q_j y_j, \qquad u|v_1\, qy\, v_2 \longrightarrow \mu_1 \dots \mu_n \times \nu_1 \dots \nu_i \, (f \nu_{i+1} \dots \nu_j) \, \nu_{j+1} \dots \nu_m \tag{3.13}$$

Dans les règles (3.12) et (3.13), tous les $(\mu_k)_{k\in[1,n]}$ et $(\nu_k)_{k\in[1,m]}$ sont des variables appartenant à des instances de non-terminaux $u'|v' \in N$ où u' et v' sont construits

à partir des termes $(p_k x_k)_{k\in[1,n]}$ et $(q_k y_k)_{k\in[1,m]}$ respectivement. Les variables μ_1 à μ_n (resp. ν_1 à ν_m) apparaissent seulement dans la première (resp. la seconde) projection de tout non-terminal. Notons que cette instanciation est unique par construction de l'ensemble N. Elle est également toujours possible puisque chaque règle de R est par hypothèse linéaire.

Soit ρ la substitution qui associe à chaque variable de non-terminal $(u|v)_i$ le terme $(u)_i$ si $i \in [1,|u|]$ et $(v)_i$ si $i \in [|u|+1,|u|+|v|]$, et à chaque variable de non-terminal $(I^i_j)_{j\in[1,2]}$ une variable x_i. Il est clair d'après la forme des règles de G_0 que

$$I \xrightarrow[G_0]{*} s \times t \iff s\rho \xrightarrow[R_+\cup R_=]{*} \circ \xrightarrow[R_-\cup R_=]{*} t\rho. \tag{3.14}$$

Nous ne détaillerons pas la démonstration de cette observation. Remarquons que cette grammaire fonctionne de manière fort similaire à un *transducteur clos d'arbres*, qui est le formalisme utilisé par [DT85] pour engendrer les dérivations d'un système clos. L'unique différence est qu'il nous faut garder la trace des variables apparaissant dans les projections gauche et droite de la relation afin de pouvoir redémarrer la réécriture aux positions correctes. Ajoutons maintenant à G_0 l'ensemble de règles

$$\forall\, x \in X \text{ tel que } xRpx,\, qxRx, \qquad px|qx \longrightarrow I \tag{3.15}$$

$$px| \longrightarrow I' \tag{3.16}$$

et appelons cette nouvelle grammaire G. Ces dernières règles permettent à la dérivation de continuer correctement après qu'une première séquence de réécriture suffixe a été effectuée, en créant de nouvelles instances de l'axiome entre les feuilles supposées être étiquetées par la même variable. D'après le lemme 3.21, G engendre $\xrightarrow[R]{*}$. □

Remarque 3.24. Remarquons que la grammaire construite dans la preuve précédente est conforme à la définition des grammaires de transduction (cf. définition 3.12. Les dérivations des systèmes suffixes sont donc une sous-famille des transductions rationnelles.

3.2.3.4 Préservation de la régularité

Contrairement aux systèmes descendants, les dérivations des systèmes suffixes préservent la régularité des langages de terme, à la fois par application directe et inverse. Ceci peut être vu comme une conséquence de la remarque 3.24, mais nous fournissons néanmoins une construction directe.

Proposition 3.25. *L'image et l'image inverse d'un langage régulier de termes par la dérivation d'un système de réécriture suffixe linéaire reconnaissable sont régulière.*

Démonstration. Soit R un système suffixe linéaire reconnaissable sur $T(F)$, G la grammaire engendrant sa dérivation telle que construite dans la démonstration du théorème 3.23, et $N \cup \{I,I'\}$ l'ensemble de non-terminaux de G. Soit A un automate fini descendant acceptant un langage régulier de termes L. Supposons que Q_A est l'ensemble d'états de contrôle de A, disjoint de F et de X, et que $q_0 \in Q_A$ est son unique état initial. Soit Q'_A une copie disjointe de Q_A, on définit la grammaire G_A ayant pour non-terminaux $Q_A \cup Q'_A \cup \{(r_1,u_1)\dots(r_n,u_n)|v \mid r_1,\dots,r_n \in Q_A \wedge u_1\dots u_n|v \in N\}$ et possédant l'ensemble de productions suivant :

– Pour toute transition $rf \underset{A}{\longrightarrow} fr_1\dots r_n$:

$$r \longrightarrow f(r_1)_1\dots(r_n)_1 \times f(r_1)_2\dots(r_n)_2 \tag{3.17}$$
$$r' \longrightarrow fr'_1\dots r'_n \tag{3.18}$$

– Pour tous $r \in Q_A$, $px|px \in N$:

$$r \longrightarrow (r,px)|px, \tag{3.19}$$

– Pour toute règle $p_1x_1\dots p_nx_n|v \longrightarrow p'_1x'_1\dots p'_nx'_n|v'$ de type (3.11) dans G et mot d'états de contrôle $r_1\dots r_n \in Q_A^*$:

$$(r_1,p_1x_1)\dots(r_n,p_nx_n)|v \longrightarrow (r_1,p'_1x'_1)\dots(r_n,p'_nx'_n)|v' \tag{3.20}$$

– Pour toute règle $u_1\,px\,u_2|v \longrightarrow s \times t$ de type (3.12) dans G avec $u_1 = p_1x_1\dots p_ix_i$, $u_2 = p_{j+1}x_{j+1}\dots p_nx_n$ et pour tout mot d'états de contrôle $r_1\dots r_i\,r\,r_{j+1}\dots r_n \in Q_A^*$:

$$(r_1,p_1x_1)\dots(r_i,p_ix_i)(r,px)(r_j,p_jx_j)\dots(r_n,p_nx_n)|v \longrightarrow s' \times t' \tag{3.21}$$

où $rf \underset{A}{\longrightarrow} fr_{i+1}\dots r_j$ et s' et t' sont formés en remplaçant dans s et t chaque occurrence de p_k dans une variable de non-terminal par (r_k,p_k), pour tout $k \in [1,n]$.

– Pour toute règle $p_1x_1\dots p_nx_n|v \longrightarrow s \times t$ de type (3.13) dans G et mot $r_1\dots r_n \in Q_A^n$:

$$(r_1,p_1x_1)\dots(r_n,p_nx_n)|v \longrightarrow s' \times t' \tag{3.22}$$

où s' et t' sont formés en remplaçant dans s et t chaque occurrence de p_k dans une variable de non-terminal par (r_k,p_k), pour tout $k \in [1,n]$.

– Enfin, pour toute règle de la forme $px|qx \longrightarrow I$ (resp. $px| \longrightarrow I'$) dans G et tout $r \in Q_A$:

$$(r,px)|qx \longrightarrow r \tag{3.23}$$
$$(r,px)| \longrightarrow r'. \tag{3.24}$$

On peut observer que, partant du non-terminal q_0, la grammaire G_A engendre exactement l'ensemble des couples (s,t) tels que $s \xrightarrow[R]{*} t$ et $s \in L$. Donc l'ensemble de tous les t tels que (s,t) est engendré par G_A depuis q_0 est exactement l'image de L par la dérivation de R :

$$L(G_A,q_0) \ = \xrightarrow[R]{*} (L).$$

Un automate reconnaissant ce langage de termes peut être construit en prenant la projection à droite de G_A, et en traitant chacune de ses variables de non-terminaux comme un non-terminal unaire. Les transitions de cet automate sont données par les règles de G_A découpées en plusieurs règles sur ces non-terminaux d'arité 1.

Remarquons que cette construction est parfaitement symétrique, et que la synchronisation de G par un automate fini A aurait pu être effectuée sur la seconde projection au lieu de la première, d'où le résultat réciproque. □

3.3 Récapitulatif et prolongements possibles

Ce travail étend les systèmes de réécriture de mots gauches, droits, préfixes et suffixes définis dans [Cau00] aux systèmes de réécriture de termes descendants, ascendants, suffixes et préfixes. Les relations de dérivation des trois premiers types de systèmes cités peuvent être engendrées par des grammaires de graphes finies, tandis que les systèmes du quatrième type ont une relation de dérivation non récursive en général. Nous établissons aussi quelques résultats de préservation de la régularité par ces systèmes, et proposons dans chaque cas des constructions effectives. Il reste cependant un certain nombre de pistes à explorer, dont nous décrivons à présent quelques-unes.

3.3.1 Classes de relations sur les termes

Cette étude propose un usage pratique de la notion de rationalité pour les relations de termes définie dans [Rao97], qui étend de façon pertinente les relations rationnelles de mots malgré leur absence de fermeture par composition ou la préservation systématique de la régularité. Cependant, ce formalisme est une base de travail intéressante et suffisamment générale pour l'étude des relations binaires sur les termes, en particulier grâce au fait qu'il est suffisamment expressif pour étendre les transducteurs asynchrones de mots (ce qui n'est pas le cas de la plupart des autres formalismes existants). Ceci dit, selon les objectifs que l'on souhaite atteindre, il pourrait être intéressant de dégager une notion plus restreinte de relations rationnelles de termes, qui soit fermée par composition ou préserve la régularité (ou les deux). Notons que [Rao97] contient la définition d'une telle

famille de relations, appelées transductions rationnelles, mais que ces relations ne contiennent pas les dérivations de tous les systèmes descendants.

Une étude plus approfondie des relations de dérivation des systèmes descendants et suffixes mérite d'être menée. En particulier, il serait intéressant de déterminer des caractérisations précises par automates de ces classes de relations.

Dans le cas des mots, les dérivations des systèmes droits (qui sont la restriction des systèmes descendants aux mots) coïncident avec la classe entière des relations rationnelles. Il semble possible de définir une classe naturelle de transducteurs d'arbres caractérisant précisément les dérivations des systèmes descendants, et dont la restriction aux mots correspond aux transducteurs classiques. L'idée principale est de considérer des ensembles gradués d'états de contrôle plutôt que des états de contrôle d'arité 1 uniquement, et de fournir des règles de transition de la forme

$$qs \to tq_1x_1 \dots q_nx_n$$

où q est un état de contrôle n-aire, s un mot de termes de longueur n, t un terme et $x_1 \dots x_n$ des mots de variables apparaissant dans s. De tels transducteurs sont syntaxiquement plus généraux que la classe des transducteurs d'arbres descendants[2]. Une condition pour que cette étude soit justifiée serait de s'intéresser également à d'autres propriétés de fermeture. Un point négatif est, comme nous l'avons déjà dit, que l'image inverse d'un langage régulier par un tel transducteur n'est pas nécessairement régulière. Une question importante reste cependant de savoir si ces relations sont closes par composition, et si cette classe peut être restreinte davantage tout en continuant à généraliser les transducteurs rationnels de mots.

Quant aux dérivations des systèmes suffixes, leur fermeture par composition est une conséquence de [Rao97], puisqu'elles appartiennent à la classe des transductions rationnelles de termes. D'autres propriétés, comme leur fermeture par les opérations booléennes, mérite d'être étudiée.

3.3.2 Graphes de réécriture descendants et suffixes

Ces résultats participent de l'étude systématique et de la compréhension de familles de relations binaires de présentation finie sur des domaines infinis, dans l'esprit des études précédentes dans le domaine des graphes infinis. En particulier, ils fournissent une définition immédiate de plusieurs familles de graphes récursifs (au sens où l'existence d'arcs est décidable), qui devraient être comparés avec soin aux familles existantes.

Les graphes de réécriture (de systèmes) descendants étendent les graphes rationnels sur les mots, ainsi que les graphes terme-synchronisés (ou terme-automa-

2. Pour de tels transducteurs, s est un terme seul et chaque x_i une variable.

tiques, cf. § 2.2.4). Une question intéressante, par exemple, serait de trouver des caractérisations différentes de cette famille de graphes, et d'étudier les langages de ses traces (pour un traitement général de ce problème dans le cas des mots, se référer au chapitre suivant).

Pour ce qui est des graphes de réécriture (des systèmes) suffixes, une question très intéressante concerne la décidabilité de leur théorie du premier ordre avec accessibilité. On sait par [DHLT90] que dans le cas des graphes définis par réécriture close, cette logique est décidable. Dans l'état de nos connaissances, il manque certains ingrédients clés pour étendre ce résultat aux systèmes suffixes, et particulièrement la fermeture par opérations booléennes des dérivations des systèmes suffixes. L'exemple suivant illustre le fait que les systèmes suffixes engendrent des graphes strictement plus généraux que les systèmes clos, ce qui fournit une motivation supplémentaire à étudier cette famille.

Exemple 3.26. Considérons le système suffixe fini $R = \{fxy \to fyx,\, a \to ga\}$ sur l'alphabet gradué $\{f^{(2)}, g^{(1)}, a^{(0)}\}$. La première règle de R permet d'échanger à tout moment d'une dérivation les deux arguments d'un opérateur f. Ceci exprime en quelque sorte la commutativité de f. La dérivation de R (restreinte pour plus de clarté à $(fg^*ag^*a)^2$) est

$$\{(fg^mag^na, fg^nag^ma) \mid m,n \geq 0\} \cup \{(fg^mag^na, fg^{m+1}ag^na) \mid m,n \geq 0\} \cup \{(fg^mag^na, fg^mag^{n+1}a) \mid m,n \geq 0\},$$

qui ne peut être engendrée par aucun transducteur clos. De plus, nous prétendons que le graphe de réécriture de ce système n'est isomorphe à aucun graphe de système clos (fini ou reconnaissable) tels qu'ils sont définis dans [Löd02, Col02]. La figure 3.3 illustre la restriction à l'ensemble (fg^*ag^*a) de ce graphe.

3.3.3 Vérification de systèmes paramétrés

Des avancées récentes dans le domaine du model checking symbolique de systèmes paramétrés s'appuient sur deux aspects fortement liés à ce travail sur les systèmes de réécriture (voir § 2.3 pour plus de détails).

Dans le cas présent, utiliser les termes comme descriptions de l'état d'un réseau paramétré arborescent et des systèmes de réécriture suffixes ou descendants comme modèles pour les transitions permet de s'assurer de l'existence d'algorithmes effectifs pour le calcul de la relation d'accessibilité du système, et par conséquent de l'image ou de l'image inverse d'un ensemble régulier quelconque. De plus, on est assuré que ces ensembles sont eux-aussi réguliers, ce qui est un point très important dans la plupart des méthodes de vérification symbolique, puisque les ensembles réguliers de termes forment une algèbre de Boole.

En terme d'expressivité, de telles règles de réécriture sont capables de modéliser des transformations complexes modifiant la structure même du réseau,

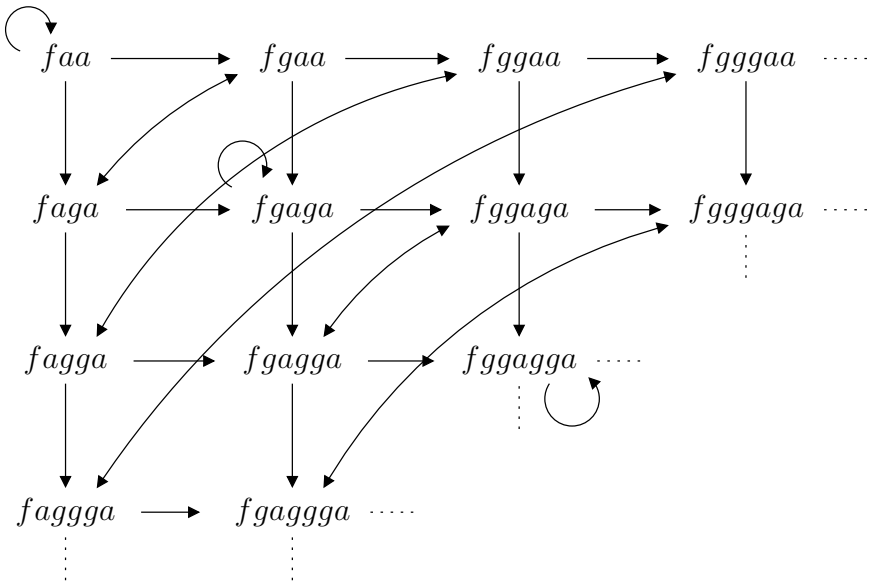


FIG. 3.3 – *Graphe de réécriture de* $R = \{fxy \to fyx, a \to ga\}$ *sur le domaine* fg^*ag^*a.

comme par exemple la création ou la destruction de processus et la modification des canaux de communication les reliant. Ce cadre met lui aussi l'accent sur l'importance d'étudier de manière rigoureuse d'autres propriétés de clôture et de décidabilité de ces systèmes de réécriture de termes.

Chapitre 4

Automates infinis pour les langages contextuels

4.1 Une hiérarchie de graphes à la Chomsky

Dans [CK02a], les auteurs définissent une hiérarchie de quatre familles de graphes infinis dont les traces sont les quatre familles de langages de la hiérarchie de Chomsky. Ils présentent ces familles de graphes, rappelées sur la figure 4.1, en utilisant le formalisme homogène des graphes de type Cayley de systèmes de réécriture de mots.

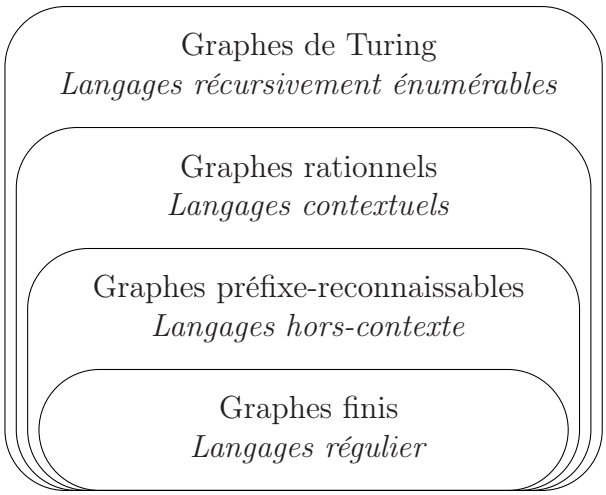


FIG. 4.1 – *Une hiérarchie à la Chomsky de graphes infinis.*

L'objectif de ce chapitre est d'approfondir l'étude de familles de graphes dont les traces sont les langages contextuels. Le paragraphe 4.2 donne une nouvelle formulation de résultats existants sur les graphes rationnels. Le paragraphe 4.3 présente une nouvelle famille de graphes plus proches des accepteurs classiques des langages contextuels, à savoir les machines de Turing linéairement bornées. Dans les deux cas, nous portons un intérêt particulier à la question du déterminisme, ainsi qu'à des considérations sur la structure de ces graphes vus comme des automates infinis. Dans le paragraphe 4.4 nous comparons ces deux familles de graphes. Nous concluons cette étude au paragraphe 4.5 par la proposition d'une autre hiérarchie de graphes infinis dont les traces coïncident avec la hiérarchie de Chomsky, dont les graphes ont cette fois un degré sortant borné et un unique sommet initial.

4.2 Graphes rationnels et leurs sous-familles

Le premier résultat sur des familles de graphes infinis dont les traces sont les langages contextuels est dû à Morvan et Stirling [MS01]. Ils ont montré que les langages acceptés par les graphes rationnels sont précisément les langages contextuels. Ce résultat fut ensuite étendu par Rispal aux familles plus restreintes des graphes rationnels *synchronisés*, et même aux graphes *synchrones* [Ris02]. Ces deux travaux utilisent des grammaires contextuelles sous une forme normale particulière définie par Penttonen [Pen74] pour caractériser les langages contextuels, ce qui pose deux problèmes. Le premier est que la forme normale de Penttonen est loin d'être évidente, et que les démonstrations et constructions fournies dans [Pen74] sont reconnues difficiles. Deuxièmement, et plus important, il n'existe pas de caractérisation par grammaires des langages contextuels déterministes, ce qui interdit l'adaptation de ces résultats au cas déterministe. Ce point particulier fut traité dans des travaux antérieurs par Carayol [Car01] et par l'auteur [Mey02] avec le souci explicite de ne pas utiliser la forme normale de Penttonen. Le but de ce paragraphe est de donner une présentation uniforme et améliorée de ces constructions.

Dans un premier temps, nous considérons le cas général où l'ensemble de sommets initiaux est énoncé comme un langage régulier de mots. Dans ce cadre, nous ré-établissons et étendons les résultats précédemment cités sans utiliser de grammaires. Notre approche utilise les systèmes de pavage (ou systèmes de tuiles), qui sont des accepteurs finis pour les langages contextuels introduits dans [LS97]. Comme il est rappelé dans l'annexe A, les systèmes de pavage sont, contrairement aux grammaires sous forme normale de Penttonen, syntaxiquement équivalents aux machines linéairement bornées (LBM). En particulier, ils admettent une notion naturelle de déterminisme qui correspond à celle des LBM. Après avoir

montré que tout graphe rationnel est trace-équivalent à un graphe rationnel à transducteurs synchrones (ou lettre-à-lettre), nous démontrons le rapport étroit qui existe entre ces graphes et les systèmes de pavage. De plus, nous montrons qu'il est possible d'accepter tout langage contextuel en se restreignant à des transducteurs synchrones consommant leur entrée de façon déterministe (aussi appelés transducteurs synchrones séquentiels).

Comme les graphes synchronisés, à la différence des graphes rationnels, ont une théorie au premier ordre décidable [BG00], ces résultats peuvent laisser penser qu'ils constituent la famille d'accepteurs infinis la plus adaptée pour les langages contextuels. Cependant, lorsqu'on considère uniquement des graphes dont le degré sortant et le nombre de sommets initiaux sont finis, les graphes rationnels synchronisés acceptent uniquement la famille des langages reconnaissables par des LBM en un nombre de renversements linéaire du sens de déplacement de la tête de lecture, que l'on peut raisonnablement supposer former une sous-famille stricte des langages contextuels. Nous donnons aussi des résultats partiels concernant les classes de langages obtenus en bornant le degré sortant des graphes. Enfin, nous montrons comment ces nouvelles constructions peuvent être utilisées pour approcher des notions satisfaisantes de déterminisme pour les graphes infinis vus comme automates : nous définissons une sous-famille syntaxique de graphes rationnels qui acceptent précisément les langages contextuels déterministes.

Ce paragraphe est structuré comme suit. Quelques définitions supplémentaires concernant les langages contextuels sont données au paragraphe 4.2.1. Les résultats concernant les langages acceptés par les graphes rationnels et leurs sous-familles apparaissent au paragraphe 4.2.2. Au paragraphe 4.2.3, nous étudions des restrictions structurelles de ces familles de graphes. Enfin, le paragraphe 4.2.4 est consacré à l'étude du cas des langages contextuels déterministes et des familles de graphes associées.

Certains des résultats présentés sont résumés dans le tableau 4.1. Un symbole d'égalité indique que les langages acceptés par la famille de graphes considérée (ligne) depuis un certain ensemble de sommets initiaux (colonne) sont précisément les langages contextuels. Un symbole d'inclusion indique que leurs langages sont strictement inclus dans les langages contextuels. Un point d'interrogation désigne une conjecture non démontrée. À chaque fois que cela se justifie, une référence à la proposition ou au théorème correspondant est donnée. Les résultats redondants et les corollaires sont omis.

4.2.1 D'autres accepteurs pour les langages contextuels

Ce paragraphe présente deux familles alternatives d'accepteurs pour les langages contextuels, à savoir les systèmes de pavage et automates cellulaires bornés. Nous donnons la définition de chacun de ces formalismes et établissons une

	Ens. rat.	Ens. i^*	Som. unique	Som. unique (d° fini)
Rationnels [MS01]	=	=	=	= [4.28]
Synchronisés [Ris02]	=	=	= [4.23]	$\subset$ (?) [4.32]
Synchrones [Ris02]	=	= [4.17]	$\subset$ [4.24]	$\subset$
Séqu. synchrones	= [4.20]	$\subset$ (?) [4.40]	$\subset$	$\subset$

TAB. 4.1 – *Familles de graphes rationnels et leurs langages.*

propriété d'équivalence de ces deux familles d'accepteurs avec les machines linéairement bornées. Dans ce chapitre, afin de simplifier notre présentation, nous ne considérerons que des langages contextuels ne contenant pas le mot vide ε (il s'agit d'une restriction courante).

4.2.1.1 Systèmes de pavage

Un formalisme méconnu permettant d'accepter les langages contextuels est celui des *systèmes de pavage* à bordures. Les systèmes de pavage ont originalement été définis pour reconnaître ou spécifier des langages d'*images*, c'est à dire de mots bi-dimensionnels sur un alphabet fini [GR96]. De tels ensembles d'images sont appelés langages *locaux* d'images. Cependant, si l'on s'intéresse uniquement à l'ensemble des mots contenus dans la première ligne de chaque image d'un langage local d'images, on obtient un langage contextuel. Réciproquement, tout langage contextuel peut être vu comme l'ensemble des premières lignes d'un langage local d'images.

Une image p de dimensions (n,m) sur un alphabet Γ est un tableau bi-dimensionnel de lettres de Γ possédant n lignes et m colonnes. On note $p(i,j)$ la lettre apparaissant à l'intersection de la i-ème ligne et de la j-ème colonne dans p (en partant du coin supérieur gauche). On note $\Gamma^{n,m}$ l'ensemble des images de dimension (n,m) et Γ^{**} l'ensemble de toutes les images[1].

Étant donné une image p de dimensions (n,m) sur Γ et une lettre $\# \notin \Gamma$, on note $p_\#$ l'image de dimensions $(n+2,m+2)$ sur $\Gamma \cup \{\#\}$ définie par :

- $p_\#(i,1) = p_\#(i,m+2) = \#$ pour tout $i \in [1,n+2]$,
- $p_\#(1,j) = p_\#(n+2,j) = \#$ pour tout $j \in [1,m+2]$,
- $p_\#(i+1,j+1) = p(i,j)$ pour tous $i \in [1,n]$ et $j \in [1,m]$.

1. On ne considère pas l'image vide.

Pour tous $n,m \geq 2$ et toute image p de dimensions (n,m), $T(p)$ désigne l'ensemble des images de dimension (2,2) apparaissant dans p. Une image de dimension (2,2) est aussi appelée *tuile*. Un langage d'images $K \subseteq \Gamma^{**}$ est dit *local* s'il existe un symbole $\# \notin \Gamma$ et un ensemble fini de tuiles Δ tels que $K = \{p \in \Gamma^{**} \mid T(p_{\#}) \subseteq \Delta\}$. À tout ensemble d'images sur Γ, on peut associer un langage de mots en isolant la frontière de chaque image. La frontière d'une image p de dimensions (n,m) est le mot $\mathrm{fr}\,(p) = p(1,1)\ldots p(1,m)$ correspondant à la première ligne de l'image.

Définition 4.1. Un système de pavage S est un quadruplet $(\Gamma,\Sigma,\#,\Delta)$ où Γ est un alphabet fini, $\Sigma \subset \Gamma$ est l'alphabet d'entrée, $\# \notin \Gamma$ est un symbole de cadre et Δ est un ensemble fini de tuiles sur $\Gamma \cup \{\#\}$. Il reconnaît le langage d'images $P(S) = \{p \in \Gamma^{**} \mid T(p_{\#}) \subseteq \Delta\}$ et le langage de mots associé $L(S) = \mathrm{fr}\,(P(S)) \cap \Sigma^*$.

Un système de pavage S reconnaît un langage $L \subseteq \Sigma^+$ en hauteur f pour une certaine application $f : \mathbb{N} \mapsto \mathbb{N}$ si pour tout $w \in L(S)$ il existe une image p de dimensions (n,m) dans $P(S)$ telle que $w = \mathrm{fr}\,(p)$ et $n \leq f(m)$.

Il n'existe pas de notion standard de déterminisme pour les systèmes de pavage utilisés comme accepteurs de langages de mots. Cependant, on peut caractériser une classe de systèmes de pavage dont les langages sont précisément les langages contextuels déterministes. On les appellera systèmes de pavage *déterministes*, à défaut d'un meilleur terme.

Définition 4.2. Un système de pavage $S = (\Gamma,\Sigma,\#,\Delta)$ est déterministe si, pour tout mot $u \in \#\Gamma^*\#$, il existe au plus un mot $v \in \#\Gamma^*\#$ tel que $|v| = |u|$ et l'image p de lignes u et v a toutes ses tuiles dans Δ.

Ceci signifie que l'on peut déterminer de façon déterministe toute ligne dans une image en fonction de la ligne précédente.

Exemple 4.3. La figure 4.2 montre l'ensemble des tuiles Δ d'un système de pavage S sur $\Gamma = \{a,b,\perp\}$, $\Sigma = \{a,b\}$ avec le symbole de cadre #. Le langage $L(S)$ est précisément le langage $\{a^n b^n \mid n \geq 1\}$. La figure 4.3 montre un élément p de $P(S)$ ainsi que l'image encadrée correspondante $p_{\#}$. Notons que dans ce cas, $T(p_{\#}) = \Delta$. Remarquons également que ce système de pavage est déterministe.

4.2.1.2 Automates cellulaires

Une troisième famille de modèles acceptant les langages contextuels, après les LBM et les systèmes de pavages, est celle des automates cellulaires travaillant en espace borné. À l'instar des systèmes de pavage, ils ne sont pas traditionnellement étudiés comme des accepteurs de langages. Cependant, in peut voir un automate cellulaire comme un une machine opérant sur des configurations. Un mot est accepté si, partant d'une certaine configuration initiale, l'automate peut la transformer en un nombre fini d'étapes en une configuration acceptante. Par

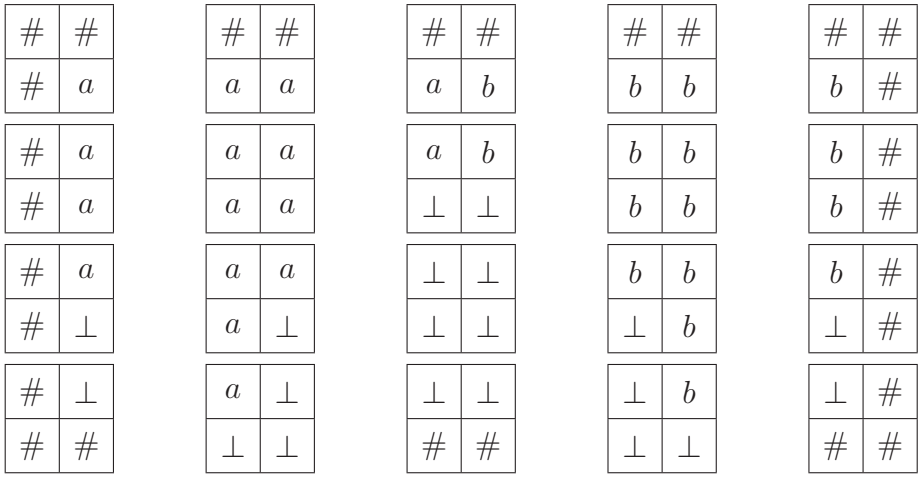


FIG. 4.2 – *Un système de pavage acceptant* $\{a^n b^n \mid n \geq 1\}$ *(cf. exemple 4.3).*

a	a	a	a	b	b	b	b
a	a	a	$\perp$	$\perp$	b	b	b
a	a	$\perp$	$\perp$	$\perp$	$\perp$	b	b
a	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	b
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

#	#	#	#	#	#	#	#	#	#
#	a	a	a	a	b	b	b	b	#
#	a	a	a	$\perp$	$\perp$	b	b	b	#
#	a	a	$\perp$	$\perp$	$\perp$	$\perp$	b	b	#
#	a	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	b	#
#	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	#
#	#	#	#	#	#	#	#	#	#

FIG. 4.3 – *Une image acceptée par l'ensemble de tuiles de la figure 4.2.*

contraste avec l'approche traditionnelle [MD98], nous autorisons les automates cellulaires à être non-déterministes.

Définition 4.4. Un *automate cellulaire* C est un sextuplet $(\Gamma,\Sigma,F,[,],\delta)$ où:

- Γ et $\Sigma \subset \Gamma$ sont l'alphabet de travail et d'entrée,
- $F \subset \Gamma$ est un ensemble de symboles terminaux,
- [et] sont des symboles de bordure qui n'appartiennent pas à Γ,
- $\delta \subseteq (\Gamma \cup \{\,[\,\}) \times \Gamma \times (\Gamma \cup \{\,]\,\}) \times \Gamma$ est la relation de transition.

On utilise la représentation suivante pour une transition $(A,B,C,D) \in \delta$:

A	B	C
	D	

Assez naturellement, on dit d'un automate cellulaire qu'il est *déterministe* si, pour tout triplet (A,B,C), il existe au plus un D tel que $(A,B,C,D) \in \delta$.

Une configuration[2] est un mot de la forme $[u]$, où $u \in \Gamma^+$. L'automate cellulaire C définit une relation de succession sur les configurations : $c' = [v]$ est un successeur de $c = [u]$ si $|c| = |c'| = n$ et pour tout $i \in [2,n-1]$, $(c(i-1),c(i),c(i+1),c'(i)) \in \delta$. Une *configuration initiale* est une configuration $[w]$ avec $w \in \Sigma^+$ et une *configuration finale* est une configuration de la forme $[F^+]$. On dit que le mot w est accepté en n étapes par C s'il existe une séquence $c_0,c_1,\ldots c_n$ telle que $c_0 = [w]$, c_n est une configuration finale et chaque c_i est successeur de c_{i-1}, pour $i \in [1,n]$.

Exemple 4.5. La figure 4.4 montre l'ensemble de règles δ d'un automate cellulaire déterministe $C = (\Gamma,\Sigma,\{\perp\},[,],\delta)$ qui peut transformer toute configuration de la forme $[a^n \perp^k b^m]$ avec $m,n > 0$ (et seulement celles-là) en $[a^{n-1} \perp^{k+2} b^{m-1}]$. Cet automate cellulaire ne peut atteindre une configuration acceptante que si $m = n$, par conséquent le langage qu'il accepte est $\{a^n b^n \mid n \geq 1\}$.

4.2.1.3 Relations entre les accepteurs

Dans [Kur64], Kuroda montra que les langages acceptés par les machines de Turing linéairement bornés sont les mêmes que les langages engendrés par des grammaires croissantes, à savoir les langages contextuels. Latteux et Simplot ont montré dans [LS97] que c'était également le cas des systèmes de pavages vus comme accepteurs de langages de mots[3]. Un résultat similaire concernant les automates cellulaires fait partie du folklore. Pour convaincre le lecteur que tous ces accepteurs sont en fait syntaxiquement équivalents, et que le passage de l'un

2. Rappelons que nous ne considérons que des langages qui ne contiennent pas le mots vide.

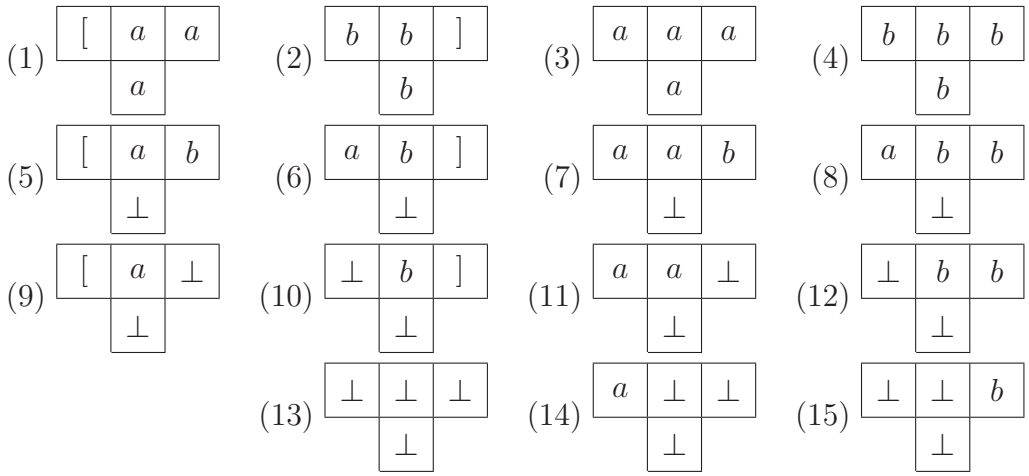


FIG. 4.4 – *Un automate cellulaire acceptant* $\{a^n b^n \mid n \geq 1\}$ *(cf. exemple 4.5).*

à l'autre ne pose pas de difficulté, des preuves complètes de ces résultats sont données en annexe A.

Théoreme 4.6. *Les simulations suivantes lient les machines de Turing linéairement bornées, les systèmes de pavage et les automates cellulaires :*

1. *Une machine de Turing linéairement bornée T travaillant en $f(n)$ renversements de la tête de lecture peut être simulée par un automate cellulaire travaillant en temps $f(n) + 2$.*
2. *Un automate cellulaire C travaillant en temps $f(n)$ peut être simulé par un système de pavage S de hauteur $f(n)$.*
3. *Un système de pavage de hauteur $f(n)$ peut être simulé par une machine de Turing linéairement bornée T travaillant en $f(n)$ renversements de tête de lecture.*

Ce résultat n'est guère surprenant étant donnée la grande ressemblance entre chacun de ces formalismes. Il est intéressant de remarquer qu'une version déterministe de ce théorème peut aussi être établie.

Théoreme 4.7. *Pour tout langage L, les propositions suivantes sont équivalentes :*

1. *L est un langage contextuel déterministe,*
2. *L est reconnu par une machine de Turing linéairement bornée déterministe,*
3. *L est reconnu par un automate cellulaire déterministe,*

3. La construction donnée par Latteux et Simplot a une borne de complexité plus élevée que celle que nous donnons au théorème 4.6. La construction correspondante est donnée en annexe (voir la proposition A.4).

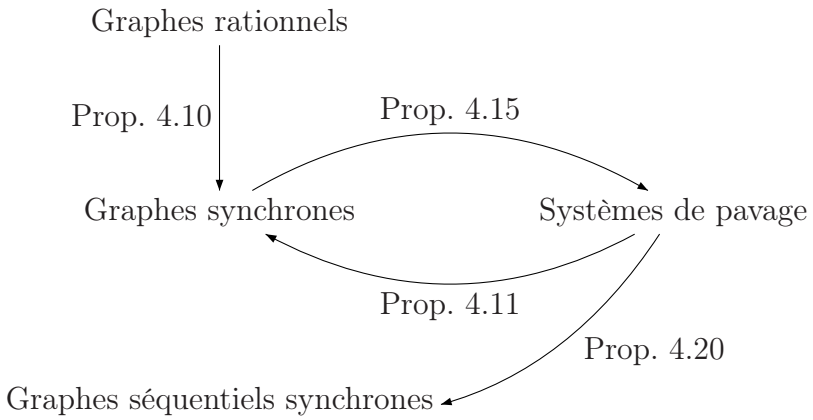


FIG. 4.5 – *Chaque arc représente une transformation préservant les langages.*

4. L est reconnu par un système de pavage déterministe.

À partir de maintenant, nous serons toujours en mesure de choisir à tout moment le type d'accepteur le plus adapté à une situation donnée.

4.2.2 Les langages des graphes rationnels

Dans ce paragraphe, nous considérons les langages acceptés par les graphes rationnels et leurs sous-familles entre deux ensembles rationnels de sommets. Nous donnons une présentation simplifiée du résultat de Morvan et Stirling [MS01] établissant que la famille des graphes rationnels accepte les langages contextuels. Ceci est effectué en plusieurs étapes. Tout d'abord, la proposition 4.10 affirme que les graphes rationnels sont trace-équivalents aux graphes rationnels *synchrones*. Ensuite, les propositions 4.11 et 4.15 établissent une relation très étroite entre graphes rationnels synchrones et systèmes de pavage. Il s'ensuit que les langages des graphes rationnels synchrones sont également les langages contextuels (théorème 4.17). Le résultat original est donné comme corollaire 4.18. Enfin, la proposition 4.20 établit que même la famille la plus restreinte que nous considérons, celle des graphes rationnels séquentiels synchrones, accepte tous les langages contextuels. Les diverses transformations présentées ici sont résumées dans la figure 4.5.

4.2.2.1 Des graphes rationnels aux graphes synchrones

Nous présentons une construction effective transformant tout graphe rationnel G muni de deux ensembles rationnels I et F de sommets initiaux et finaux en un graphe rationnel synchrone G' trace-équivalent entre deux ensembles rationnels

I' et F'. L'idée de la construction est de remplacer le symbole ε apparaissant dans les transitions des transducteurs définissant G par un nouveau symbole visible $\#$.

Soit $(T_a)_{a\in\Sigma}$ l'ensemble des transducteurs sur Γ caractérisant G et soit $\#$ un symbole hors de Γ. Pour tout A, on définit $\bar{A}$ comme valant A si $A \in \Gamma$, et ε si $A = \#$. On étend par morphisme cette application en une projection de $(\Gamma\cup\#)^*$ vers Γ^*. On définit G' comme le graphe rationnel défini par l'ensemble de transducteurs $(T'_a)_{a\in\Sigma}$ où T'_a a les mêmes états de contrôle Q_a que T_a et l'ensemble de transitions

$$\left\{ p \xrightarrow{A/B} q \mid p \xrightarrow{\bar{A}/\bar{B}} q \in T_a \right\} \cup \left\{ p \xrightarrow{\#/\#} p \mid p \in Q_a \right\}.$$

Par définition de chaque T'_a, G' est un graphe rationnel synchrone. Soient I' et F' deux ensembles rationnels tels que $I' = \{u \mid \bar{u} \in I\}$ et $F' = \{v \mid \bar{v} \in F\}$ (l'automate acceptant I' (resp. F') est obtenu à partir de l'automate acceptant I (resp. F) en ajoutant une boucle étiquetée par $\#$ à chacun de ses états de contrôle). Nous affirmons que G' accepte bien le même langage entre I' et F' que G entre I et F. Par exemple, la figure 4.6 illustre cette construction appliquée au graphe de la figure 2.6. Seule une composante connexe du graphe est représentée.

Avant de démontrer la correction de cette construction, nous avons besoin d'établir deux lemmes techniques. Soit $\mathcal{B}$ l'ensemble des applications de $\mathbb{N}$ dans $\mathbb{N}$. À chaque application $\delta \in \mathcal{B}$, nous associons une application de $(\Gamma \cup \{\#\})^*$ dans $(\Gamma \cup \{\#\})^*$ définie comme suit : pour tout $w = \#^{i_0}A_1\#^{i_1}\dots A_n\#^{i_n}$ avec $A_1,\dots,A_n \in \Gamma$, on pose $\delta w = \#^{i_0+\delta(0)}A_1\#^{i_1+\delta(1)}\dots A_n\#^{i_n+\delta(n)}$. Avant de continuer, nous établissons deux propriétés de ces applications par rapport aux ensembles de transducteurs (T_a) et (T'_a).

Lemme 4.8. $\forall u,v \in \Gamma^*,\ (u,v) \in T_a \iff \exists \delta_u,\delta_v \in \mathcal{B},\ (\delta_u u,\delta_v v) \in T'_a$.

Démonstration. Nous commençons par montrer que pour tout chemin ρ étiqueté par u/v entre les états p et q de T_a, il existe un couple d'applications δ_u, δ_v tel qu'il existe un chemin dans T'_a étiqueté par $\delta_u u/\delta_v v$ entre p et q. Ceci peut être fait par récurrence sur la longueur de ρ. Si $|\rho| = 0$, on peut simplement prendre $\delta_u(i) = \delta_v(i) = 0$ pour tout i. Sinon, si $\rho = k$ avec $k > 0$, il doit exister un état p' tel que ρ s'écrit $p \xrightarrow{x/y} p' \xrightarrow{u'/v'} q$ avec $x,y \in \Gamma \cup \{\varepsilon\}$ et $u',v' \in \Gamma^*$. Par hypothèse de récurrence, on peut trouver deux applications $\delta_{u'}$ et $\delta_{v'}$ telles que $\delta_{u'}u'/\delta_{v'}v'$ étiquette un chemin entre p' et q dans T'_a. Par construction, il existe une transition $p \xrightarrow{x'/y'} p'$ dans T'_a telle que $x = \bar{x}'$ et $y = \bar{y}'$. Donc pour

$$\delta_u = \begin{cases} \{0 \mapsto 0\} \cup \{i \mapsto \delta_{u'}(i-1) \mid i > 0\} & \text{if } x \neq \varepsilon \\ \{0 \mapsto \delta_{u'}(0)+1\} \cup \{i \mapsto \delta_{u'}(i) \mid i > 0\} & \text{if } x = \varepsilon \end{cases}$$

et δ_v défini de façon similaire par rapport à $\delta_{v'}$ et y, on a $p \xrightarrow{\delta_u u/\delta_v v} q \in T'_a$.

Réciproquement, s'il existe un chemin dans T'_a étiqueté par u'/v', alors par construction il doit aussi exister un chemin dans T_a étiqueté par $\bar{u}'/\bar{v}'$. Donc pour tout couple d'applications δ_u, δ_v tel que $\delta_u\bar{u}' = u'$ et $\delta_v\bar{v}' = v'$, la propriété est vérifiée. □

Nous établissons maintenant une seconde propriété des applications de $\mathcal{B}$.

Lemme 4.9. $\forall(u,v) \in T'_a$, $\forall\delta \in \mathcal{B}$, $\exists\delta' \in \mathcal{B}$, $(\delta u,\delta' v) \in T'_a$,
et réciproquement $\forall(u,v) \in T'_a$, $\forall\delta \in \mathcal{B}$, $\exists\delta' \in \mathcal{B}$, $(\delta' u,\delta v) \in T'_a$.

Démonstration. Nous allons démontrer une propriété légèrement plus générale. Considérons un chemin ρ quelconque étiqueté par un couple u/v entre deux états de contrôle p et q dans le transducteur T'_a, et une application $\delta \in \mathcal{B}$. Notre but est de définir une seconde application δ' telle qu'il existe un chemin entre p et q dans T'_a étiqueté par $\delta u/\delta' v$. L'existence de δ' peut être démontrée par récurrence sur la longueur de ρ.
Cas de base. Comme T'_a est synchrone et $|\rho| = 0$, on a $u = v = \varepsilon$ et $p = q$. Si l'on prend $\delta'(0) = \delta(0)$ et $\delta'(i) = 0$ pour tout $i > 0$, alors $\delta u = \delta' v = \#^{\delta(0)}$. Ce chemin peut être obtenu dans T'_a en empruntant $\delta(0)$ fois la boucle $\#/\#$ sur p.
Étape de récurrence. Supposons la propriété vraie pour tout chemin de taille inférieure ou égale à n, et soit ρ un chemin de longueur n dans T'_a étiqueté par u/v entre les états p et q. Par hypothèse de récurrence, pour tout δ il existe une application δ' telle que $\delta u/\delta' v$ étiquette aussi un chemin entre p et q. Considérons maintenant l'ajout à la fin du chemin ρ d'une transition étiquetée par x/y entre q et un certain état q'. Nous allons définir une application δ'' à partir de δ' en fonction des valeurs de x et y. Quatre cas doivent être considérés :

1. Si $x = \#$, $y = \#$, $\quad \delta'' = \delta'$
2. Si $x \in \Gamma$, $y \in \Gamma$, $\quad \delta''(i) = \begin{cases} \delta'(i) & \text{si } i \leq |\bar{v}| \\ \delta(|\bar{u}|+1) & \text{si } i = |\bar{v}|+1 \end{cases}$
3. Si $x \in \Gamma$, $y = \#$, $\quad \delta''(i) = \begin{cases} \delta'(i) & \text{si } i < |\bar{v}| \\ \delta'(i) + \delta(|\bar{u}|+1) & \text{si } i = |\bar{v}| \end{cases}$
4. Si $x = \#$, $y \in \Gamma$, $\quad \delta''(i) = \begin{cases} \delta'(i) & \text{si } i \leq |\bar{v}| \\ 0 & \text{si } i = |\bar{v}|+1 \end{cases}$

Quand elle n'est pas précisée, la valeur de $\delta''(i)$ est supposée nulle. Nous affirmons que cette définition assure que $\delta ux/\delta'' vy$ étiquette un chemin valide dans T'_a entre p et q', ce qui conclut la démonstration par récurrence. Dans le cas où ρ est acceptant, nous obtenons la preuve de la première propriété. Le cas duel est démontré de façon similaire. □

Nous pouvons maintenant démontrer la correction de la construction : un mot w est accepté par G entre I et F si et seulement si il est accepté par G' entre I' et F'.

Proposition 4.10. *Pour tout graphe rationnel G et tous ensembles rationnels de sommets I et F, il existe un graphe rationnel synchrone et deux ensembles rationnels I' et F' tels que $L(G,I,F) = L(G',I',F')$.*

Démonstration. Nous montrons par récurrence sur n que pour tout $u_0, \ldots, u_n \in \Gamma^*$, s'il existe un chemin

$$u_0 \xrightarrow[G]{w(1)} u_1 \ldots u_{n-1} \xrightarrow[G]{w(n)} u_n,$$

alors il doit exister des mots $u'_0, \ldots, u'_n \in (\Gamma \cup \{\#\})^*$ tels que pour tout i, $\bar{u}'_i = u_i$ et

$$u'_0 \xrightarrow[G']{w(1)} u'_1 \ldots u'_{n-1} \xrightarrow[G']{w(n)} u'_n.$$

Le cas où $n = 0$ est trivial. Supposons la propriété vraie pour tout chemin de taille inférieure ou égale à n, et considérons un chemin

$$u_0 \xrightarrow[G]{w(1)} \ldots \xrightarrow[G]{w(n)} u_n \xrightarrow[G]{w(n+1)} u_{n+1}.$$

Par hypothèse de récurrence, on peut trouver des applications $\delta_0, \ldots \delta_n$ telles que

$$\delta_0 u_0 \xrightarrow[G']{w(1)} \ldots \xrightarrow[G']{w(n)} \delta_n u_n.$$

D'après le lemme 4.8, il existe δ'_n et δ'_{n+1} tels que $\delta'_n u_n \xrightarrow[G']{w(n+1)} \delta'_{n+1} u_{n+1}$. Soient γ_n et γ'_n deux éléments de $\mathcal{B}$ tels que $\delta'_n \circ \gamma'_n = \delta_n \circ \gamma_n$. D'après le lemme 4.9, on peut trouver des applications γ'_{n+1} et γ_0 jusqu'à γ_{n-1} telles que

$$\gamma_0 \delta_0 u_0 \xrightarrow[G']{w(1)} \ldots \xrightarrow[G']{w(n)} \gamma_n \delta_n u_n = \gamma'_n \delta'_n u_n \xrightarrow[G']{w(n+1)} \gamma'_{n+1} \delta'_{n+1} u_{n+1},$$

ce qui conclut la preuve par récurrence. Si l'on suppose que $u_0 \in I$ et $u_n \in F$, alors nécessairement $u'_0 \in I'$ et $u'_n \in F'$. Il s'ensuit que pour tout chemin dans G entre I et F, il existe un chemin dans G' entre I' et F' possédant la même étiquette.

Réciproquement, d'après le lemme (4.8), pour tout chemin dans G' l'effacement des occurrences de # de ses sommets produit un chemin valide dans G entre I et F. Par conséquent $L(G,I,F) = L(G',I',F')$. □

4.2.2.2 Équivalence entre graphes synchrones et systèmes de pavage

La proposition suivante établit la relation étroite existant entre les systèmes de pavage et les graphes rationnels synchrones. La proposition 4.11 présente une transformation effective d'un système de pavage en un graphe rationnel synchrone.

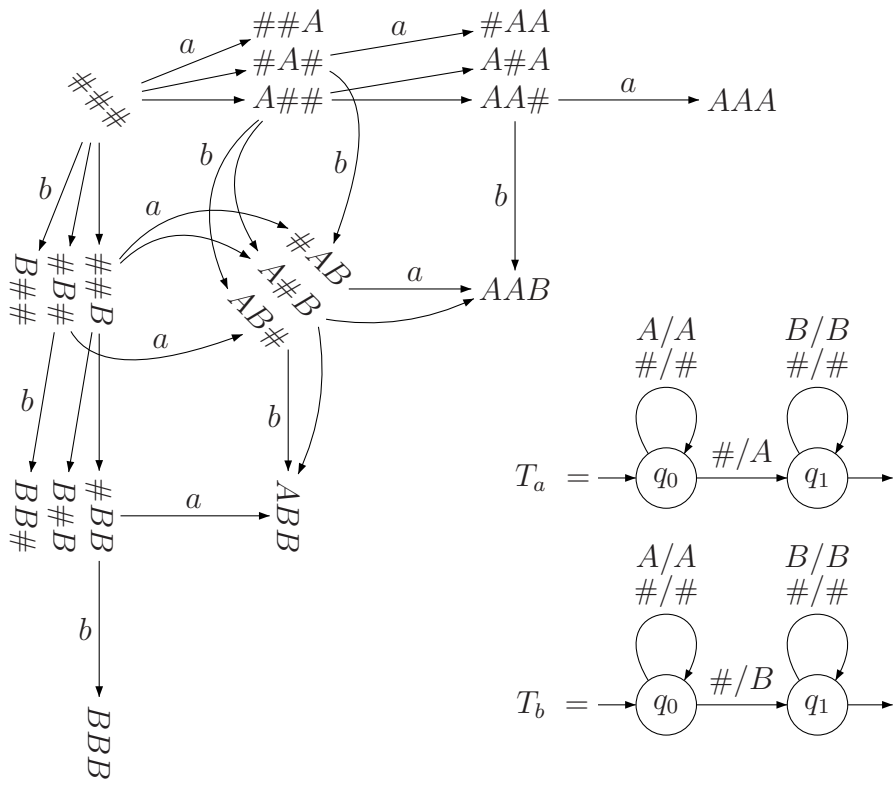


FIG. 4.6 – *Graphe rationnel synchrone trace-équivalent à la grille (une composante connexe), cf. figure 2.6 p. 62.*

Proposition 4.11. *Étant donné un système de pavage $S = (\Gamma,\Sigma,\#,\Delta)$, il existe un graphe rationnel synchrone G et deux ensembles rationnels I et F tels que $L(G,I,F) = L(S)$.*

Démonstration. Considérons l'automate fini A sur Γ d'ensemble d'états de contrôle $Q = \Gamma \cup \{\#\}$, d'état initial $\#$, d'ensemble d'états finaux F et dont l'ensemble des transitions δ est donné par :

$$F \ : \ a \quad \text{tel que} \quad \begin{array}{|c|c|}\hline a & \# \\ \hline \# & \# \\ \hline \end{array} \in \Delta$$

$$\delta \ : \ \# \xrightarrow[A]{a} a,\ a \xrightarrow[A]{b} b \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline a & \# \\ \hline \end{array}\ , \ \begin{array}{|c|c|}\hline a & \# \\ \hline b & \# \\ \hline \end{array} \in \Delta \text{ (resp.).}$$

Appelons M le langage reconnu par A, M représente l'ensemble des dernières colonnes possibles d'images de $P(S)$. Notons que cela n'implique pas que tout mot de M *est effectivement* la dernière colonne d'une image de $P(S)$, mais uniquement qu'il est compatible avec les tuiles de bordure droite de Δ.

Construisons un graphe rationnel synchrone G et deux ensembles rationnels I et F tels que $L = L(G,I,F)$. Les transitions de l'ensemble des transducteurs $(T_e)_{e\in\Sigma}$ de G sont:

$$(\#,\#) \xrightarrow[T_d]{c/d} (c,d) \qquad \text{pour tout} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline c & d \\ \hline \end{array} \in \Delta,\ d \neq \#$$

$$(a,b) \xrightarrow[T_e]{c/d} (c,d) \qquad \text{pour tout} \quad \begin{array}{|c|c|}\hline a & b \\ \hline c & d \\ \hline \end{array} \in \Delta,\ b,d \neq \#,\ e \in \Sigma$$

où $(\#,\#)$ est l'unique état initial de chaque transducteur et l'ensemble des états finaux F de chacun d'entre eux est

$$F \ : \ (a,b) \in (\Gamma \cup \{\#\}) \times \Gamma \ \text{ tels que } \begin{array}{|c|c|}\hline a & b \\ \hline \# & \# \\ \hline \end{array} \in \Delta.$$

Un couple de mots (s,t) est accepté par le transducteur T_e si et seulement si e est la première lettre de t et soit s et t sont deux colonnes adjacentes d'une image de $P(S)$, soit $s \in \#^*$ et t est la première colonne d'une image de $P(S)$. Par conséquent, $L = L(G,\#^*,M)$. □

Exemple 4.12. La figure 4.7 montre les transducteurs obtenus en appliquant cette construction au système de pavage de la figure 4.2. Ils définissent un graphe

rationnel synchrone dont le langage de chemins entre $\#^*$ et $b^*\perp$ est $\{a^nb^n \mid n \geq 1\}$, représenté sur la figure 4.8. Les sommets de ce graphe sont les mots de l'ensemble rationnel $\#^{\geq 2} \cup a^+\perp^+ \cup b^+\perp^+$, l'ensemble des sommets initiaux est $\#^{\geq 2}$ et l'ensemble des sommets finaux $b^+\perp$. Remarquons que dans cet exemple, l'ensemble des sommets accessibles depuis les sommets initiaux est rationnel, ce qui n'est pas vrai dans le cas général.

Remarque 4.13. La correspondance entre un système de pavage S et le graphe rationnel synchrone correspondant G construit à la proposition 4.11 est très étroite : chaque image p de frontière w correspond à un unique chemin acceptant pour w dans G (et réciproquement). Plus formellement, il existe une bijection ϕ entre les images de $P(S)$ et l'ensemble des chemins acceptants de G entre I et F. En particulier, pour tout $p \in P(S)$ de frontière w et de hauteur n, $\phi(p)$ est un chemin acceptant pour w dont les sommets sont de longueur n.

Pour simplifier la démonstration de la réciproque, nous commençons par démontrer que les ensembles de sommets initiaux et finaux peuvent être choisis sur un alphabet à une lettre sans perte de généralité.

Lemme 4.14. *Pour tout graphe rationnel synchrone G à sommets dans Γ^* et tous ensembles rationnels I et F, on peut trouver un graphe rationnel synchrone H et deux symboles i et $f \notin \Gamma$ tels que $L(G,I,F) = L(H,i^*,f^*)$.*

Démonstration. Soit $G = (K_a)_{a\in\Sigma}$ un graphe rationnel synchrone à sommets dans Γ^*. Pour deux nouveaux symboles distincts i et f quelconques, on définit un nouveau graphe rationnel synchrone H caractérisé par l'ensemble de transductions

$$\big(T_a = (T_I \circ K_a) \ \cup \ K_a \ \cup \ (K_a \circ T_F)\big)_{a\in\Sigma}$$

où $T_I = \{(i^n,u) \mid n \geq 0,\, u \in I,\, |u| = n\}$ et $T_F = \{(v,f^n) \mid n \geq 0,\, v \in F,\, |v| = n\}$. Pour tous sommets $u \in I$, $v \in F$ nous avons $u \xrightarrow[G]{w} v$ si et seulement si $i^{|u|} \xrightarrow[H]{w} f^{|u|}$, c'est à dire $L(G,I,F) = L(H,i^*,f^*)$. □

Nous pouvons à présent établir la réciproque de la proposition 4.11, qui affirme que tout langage accepté par un graphe rationnel synchrone entre deux ensembles rationnels de sommets peut être accepté par un système de pavage.

Proposition 4.15. *Étant donné un graphe rationnel synchrone G et deux ensembles rationnels I et F, il existe un système de pavage S tel que $L(S) = L(G,I,F)$.*

Démonstration. Soit $G = (T_a)_{a\in\Sigma}$ un graphe rationnel synchrone à sommets dans Γ^* (avec $\Sigma \subseteq \Gamma$). D'après le lemme 4.14, on peut supposer sans perte de généralité que $I = i^*$ et $F = f^*$ pour deux lettres distinctes i et f, et que ni i ni f n'apparaît dans un sommet qui ne soit pas dans I ou F. De plus d'après la remarque 1.28, on peut supposer que T_a est non-ambigu pour tout $a \in \Sigma$.

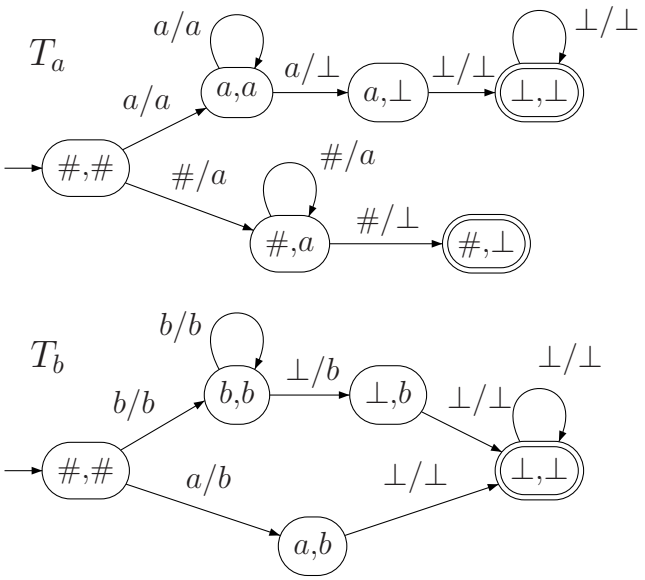


FIG. 4.7 – *Transducteurs d'un graphe synchrone acceptant* $\{a^n b^n \mid n \geq 1\}$.

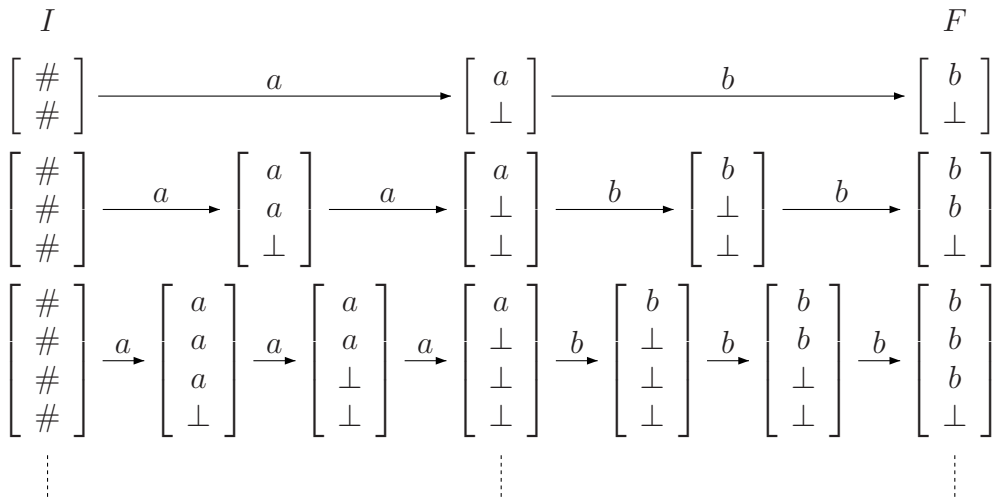


FIG. 4.8 – *Le graphe rationnel synchrone associé au système de pavage de la figure 4.2, dont les transducteurs sont présentés à la figure 4.7.*

Pour tout $a \in \Sigma$ on note Q_a l'ensemble d'états de contrôle de T_a. Supposons que tous les ensembles d'états de contrôle sont disjoints, et notons $q_0^a \in Q_a$ l'unique état initial de chaque transducteur T_a, et Q_F l'ensemble de ses états finaux.

Soient $a,b,c,d \in \Sigma$, $x,x',y,y',z,z' \in \Gamma$, et $p,p',q,q',r,r',s,s' \in \bigcup_{a\in\Sigma} Q_a$. Nous définissons le système de pavage $S = (\Gamma,\Sigma,\#,\Delta)$, où Δ est l'ensemble de tuiles suivant :

$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	
$\#$	a	b	c	d	$\#$	
$\#$	a	b	c	d	$\#$	avec $q_0^a \xrightarrow[T_a]{i/x} p$, $q_0^c \xrightarrow[T_c]{y/z} r$, $s \in Q_d$
$\#$	$x\,p$	$y\,q$	$z\,r$	$f\,s$	$\#$	
$\#$	$x\,p$	$y\,q$	$z\,r$	$f\,s$	$\#$	avec $\exists a,b,c$, $p \xrightarrow[T_a]{i/x'} p'$, $r \xrightarrow[T_b]{y'/z'} r'$, $s,s' \in Q_c$
$\#$	$x'p'$	$y'q'$	$z'r'$	fs'	$\#$	
$\#$	$x\,p$	$y\,q$	$z\,r$	$f\,s$	$\#$	avec $p,q,r,s \in Q_F$
$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	

Soit ϕ la fonction associant à chaque image $p \in P(S)$ de colonnes $a_1w_1,\ldots,a_nw_n$ le chemin $i^{|w_1|} \xrightarrow{a_1} \widetilde{w_1} \ldots \xrightarrow{a_n} \widetilde{w_n}$, où $\widetilde{w}$ est obtenu en supprimant les états de contrôle de w. Par construction de S, la fonction ϕ est bien définie. Il est facile de vérifier que ϕ est surjective. Comme les transducteurs définissant G sont non-ambigus, deux images distinctes ont une image distincte par ϕ et par conséquent ϕ est aussi injective, c'est donc une bijection.

Nous avons donc démontré que le système de pavage $(\Gamma,\Sigma,\#,\Delta)$ reconnaît exactement le langage $L(G,I,F)$. □

Remarque 4.16. Comme pour la remarque 4.13, l'ensemble des chemins de G de I à F et l'ensemble des images $P(S)$ acceptées par S sont en bijection, et la longueur des sommets le long d'un chemin est égal à la hauteur de l'image correspondante.

En combinant les propositions 4.11 et 4.15 et le théorème 4.6, on obtient le résultat suivant concernant les traces des graphes rationnels synchrones.

Théoreme 4.17 ([Ris02])**.** *Les langages acceptés par les graphes rationnels synchrones entre des ensembles rationnels de sommets initiaux et finaux sont les langages contextuels.*

Remarquons que cette formulation du théorème pourrait être rendue légèrement plus précise en rappelant que les ensembles de sommets initiaux et finaux de la forme x^*, où x est une lettre, sont en fait suffisants pour accepter tous les langages contextuels, comme il est dit au lemme 4.14.

D'après la proposition 4.10, nous retrouvons le résultat original par Morvan et Stirling [MS01].

Corollaire 4.18. *Les langages acceptés par les graphes rationnels entre des ensembles rationnels de sommets initiaux et finaux sont les langages contextuels.*

Ce résultat peut être légèrement renforcé au cas d'un sommet initial et d'un sommet final uniques.

Corollaire 4.19. *Pour tout graphe rationnel G étiqueté par Σ et tout symbole $\sharp \notin \Sigma$, le langage $L_G = \{i\sharp w\sharp f \mid w \in L(G,i,f)\}$ est contextuel.*

Démonstration. Soit G un graphe rationnel étiqueté par Σ à sommets dans Γ^* défini par une famille de transducteurs $(T_a)_{a\in\Sigma}$. Soient $\bar{\Gamma}$ et $\tilde{\Gamma}$ deux alphabets finis disjoints en bijection avec Γ. Pour tout $x \in \Gamma$, on note $\bar{x}$ (resp. $\tilde{x}$) le symbole correspondant dans $\bar{\Gamma}$ (resp. $\tilde{\Gamma}$). Nous considérons le graphe rationnel H étiqueté par $\Xi = \Sigma \cup \bar{\Gamma} \cup \tilde{\Gamma}$ défini par la famille de transducteurs $(T_x)_{x\in\Xi}$, où pour tout $x \in \Gamma$, $T_{\bar{x}} = \{(u,ux) \mid u \in \Gamma^*\}$ et $T_{\tilde{x}} = \{(xu,u) \mid u \in \Gamma^*\}$.

D'après le corollaire 4.18, le langage $L = L(H,\varepsilon,\varepsilon) \cap \bar{\Gamma}^*\Sigma^*\tilde{\Gamma}^*$ est contextuel. Par construction, L est égal à $\{\bar{i}w\tilde{f} \mid i,f \in \Gamma^* \text{ and } w \in \Sigma^*\}$. Il s'ensuit que L_G est contextuel. □

Notons que si l'on transforme un graphe rationnel en machine de Turing en appliquant successivement les constructions des propositions 4.10 et 4.15 et du théorème 4.7, on obtient la même machine de Turing que dans [MS01].

4.2.2.3 Les graphes séquentiels synchrones suffisent

Le théorème 4.17 montre que lorsque l'on considère des ensembles rationnels de sommets initiaux, les graphes synchrones suffisent à accepter tous les langages contextuels. Il est naturel de se demander s'il est possible de restreindre encore davantage la famille de graphe considérée tout en conservant cette propriété. De façon surprenante, grâce aux ensembles rationnels d'états initiaux et finaux, la classe très restreinte des graphes séquentiels synchrones suffit, comme l'exprime la proposition suivante.

Proposition 4.20. *Les langages acceptés par les graphes rationnels séquentiels synchrones entre des ensembles rationnels de sommets initiaux et finaux sont les langages contextuels.*

Démonstration. Grâce à la proposition 4.15, il suffit de montrer que tout langage contextuel $L \subseteq \Sigma^*$ est accepté par au moins un graphe rationnel séquentiel synchrone. D'après le théorème 4.6, nous savons qu'il existe un système de pavage $S = (\Gamma,\Sigma,\#,\Delta)$ tel que $L(S) = L$.

Soit $\Lambda = \Gamma \cup \{\#\}$, et soient [et] deux symboles n'appartenant pas à Λ. Nous associons à chaque image $p \in \Lambda^{**}$ ayant comme lignes $l_1,\ldots,l_n$ le mot $[l_1]\ldots[l_n]$.

Nous allons définir un ensemble de transducteurs séquentiels synchrones qui, une fois itérés, reconnaissent les mots correspondant aux images de $P(S)$.

Tout d'abord, pour tout ensemble fini de tuiles Δ, nous construisons un transducteur T_Δ qui vérifie qu'un mot dans $([\Lambda^{\geq 3}])^{\geq 2}$ représente bien une image dont les tuiles sont dans Δ. La vérification s'effectue colonne par colonne, et nous introduisons des lettres marquées pour garder la trace de la colonne en cours de vérification. Soit $\widetilde{\Lambda}$ un alphabet fini en bijection avec (mais disjoint de) Λ. Pour tout $x \in \Lambda$ on note $\widetilde{x} \in \widetilde{\Lambda}$ la version marquée x. Pour tout mot $w = u\widetilde{x}v \in \Lambda^*\widetilde{\Lambda}\Lambda^*$, on note $\pi(w)$ le mot $uxv \in \Lambda^*$ et $\rho(w) = |u| + 1$ désigne la position de la lettre marquée dans ce mot.

Nous considérons des mots dans $[\Lambda^*\widetilde{\Lambda}\Lambda^*]^{\geq 2}$. Soit Shift la relation qui décale d'un cran vers la droite toutes les marques apparaissant dans un mot. Plus précisément, Shift est telle que $Dom(\text{Shift}) = ([\Lambda^*\widetilde{\Lambda}\Lambda^+])^{\geq 2}$, et $\text{Shift}([w_1]\dots[w_n]) = [w'_1]\dots[w'_n]$ avec $\pi(w'_i) = \pi(w_i)$ et $\rho(w'_i) = \rho(w_i) + 1$ pour tout $i \in [1,n]$. La relation rationnelle Shift peut être réalisée par un transducteur séquentiel synchrone T_{Sh}. Considérons le langage rationnel

$$R_\Delta = \left\{ [w_1x_1\widetilde{y_1}w'_1]\dots[w_nx_n\widetilde{y_n}w'_n] \mid n \geq 2 \text{ and } \forall i \in [2,n], \begin{array}{|c|c|}\hline x_{i-1} & y_{i-1} \\ \hline x_i & y_i \\ \hline \end{array} \in \Delta \right\}.$$

D'après le lemme 1.29, le transducteur T_Δ obtenu en restreignant T_{Sh} au domaine R_Δ est à la fois séquentiel et synchrone. Pour tout $w = [w_1]\dots[w_n] \in ([\Lambda\widetilde{\Lambda}\Lambda^*])^{\geq 2}$, si $w' = T_\Delta^N(w)$ alors $w' = [w'_1]\dots[w'_n]$ avec $\pi(w_i) = \pi(w'_i)$ et $\rho(w'_i) = N + 2$ pour tout $i \in [1,n]$. Soit r_i le mot contenant les $N + 1$ premières lettres de w'_i, une récurrence simple sur N démontre que l'image p formée des lignes $r_1, \dots, r_n$ n'a que des tuiles dans Δ. En particulier, $T_\Delta^N(w)$ appartient à $([\Lambda^*\widetilde{\Lambda}])^* \cap R_\Delta$ si et seulement si $\pi(w)$ représente une image p de largeur $N + 2$ telle que $T(p) \subseteq \Delta$.

Nous définissons maintenant plus précisément le graphe rationnel séquentiel $G = (T_a)_{a\in\Sigma}$ acceptant L. Pour tout $a \in \Sigma$, le transducteur T_a est obtenu en restreignant le domaine de T_Δ à l'ensemble des mots représentant des images dont le symbole marqué à la deuxième ligne est a, c'est à dire à l'ensemble $[(\Lambda \cup \widetilde{\Lambda})^*][(\Lambda^*\widetilde{a}\Lambda^*][(\Lambda \cup \widetilde{\Lambda})^*]^*$. D'après le lemme 1.29, T_a peut être choisi séquentiel et synchrone. L'ensemble de sommets initiaux I est $[\#\widetilde{\#}\#^*]([\#\widetilde{\Gamma}\Gamma^*\#])^*[\#\widetilde{\#}\#^*]$ et l'ensemble de sommets finaux F est $[\#^*\widetilde{\#}]([\#\Gamma^*\widetilde{\#}])^*[\#^*\widetilde{\#}]$.

Démontrons que $L = L(G,I,F)$. Supposons que pour un certain mot $i \in I$, $T_w(i)$ appartienne à F pour un certain $w = a_1 \dots a_n$. Ceci implique que $T_\Delta^n(i)$ appartient à F et donc i représente une image p de largeur $n+2$ telle que $T(p) \subseteq \Delta$. Comme i appartient à I il existe une image $q \in \Gamma^{**}$ telle que $p = q_\#$. De plus, la frontière de q est w et donc w appartient à L. Nous avons démontré que $L(G,I,F) \subseteq L$. Réciproquement, soit $w = a_1 \dots a_n$ un mot de L, il existe une

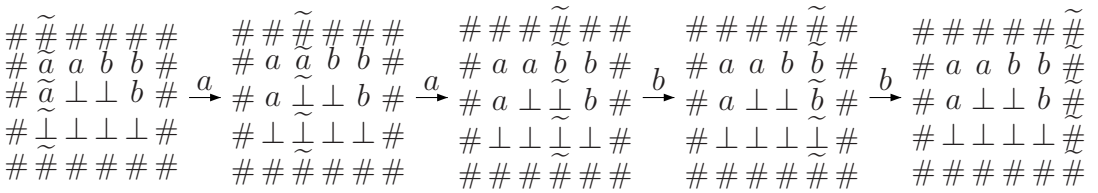


FIG. 4.9 – *Composante connexe d'un graphe rationnel séquentiel synchrone acceptant* $\{a^n b^n \mid n \geq 1\}$.

---

image q telle que $w = \mathrm{fr}\,(q)$ et $T(q_\#) \subseteq \Delta$. Soit i le mot représentant $q_\#$, il est facile de montrer que $T_w(i)$ appartient à F. Ceci démontre que $L \subseteq L(G,I,F)$. □

Exemple 4.21. La figure 4.9 montre une partie du résultat de la précédente construction appliquée au langage $\{a^n b^n \mid n \geq 1\}$ tel qu'il est reconnu par le système de pavage de la figure 4.2. Par commodité chaque sommet est représenté par l'image correspondante, plutôt que par le mot qui la code. Aussi, seule une composante connexe du graphe est montrée. Les autres composantes ont toutes la même structure linéaire : le degré du graphe est borné par 1. Le sommet le plus à gauche sur le dessin appartient à I, et le plus à droite appartient à F, et donc le mot a^2b^2 est accepté par le graphe.

Remarque 4.22. Dans le cas de transducteurs synchrones, il a été montré au lemme 4.14 que I pouvait être choisi sur un alphabet à une seule lettre sans perte de généralité. Ceci ne semble pas être le cas de transducteurs séquentiels étant donné que la démonstration que nous présentons s'appuie sur l'information contenue dans l'ensemble de sommets initiaux. En fait, comme le montre la proposition 4.40, les langages acceptés par les graphes séquentiels synchrones depuis les ensembles de la forme i^* sont des langages contextuels déterministes.

4.2.3 Graphes rationnels vus comme des automates

La structure des graphes obtenus dans le paragraphe précédent (propositions 4.11 et 4.20) est très pauvre. Les graphes synchrones sont par définition composés d'un ensemble potentiellement infini de composantes connexes *finies*. Dans le cas de la proposition 4.20, on obtient une famille de graphes encore plus restreinte puisque leur degré est borné par 1. Cependant, si l'on considère les langages acceptés d'un ensemble rationnel de sommets vers un autre ensemble rationnel, on obtient même dans ce cas extrêmement restreint autant de langages que dans le cas des graphes rationnels généraux, c'est à dire l'ensemble de tous les langages contextuels. C'est pourquoi, afin de mieux pouvoir comparer les différences d'expressivité de chaque sous-famille des graphes rationnels et afin d'obtenir des

graphes possédant une structure plus riche, nous avons besoin d'imposer des restrictions structurelles.

Dans un premier temps, on considère des graphes ne possédant qu'un unique sommet initial, mais cette restriction seule ne suffit pas. En effet, les graphes synchronisés comme les graphes rationnels à sommet initial unique acceptent la même famille de langages qu'avec un ensemble rationnel de sommets initiaux.

Lemme 4.23. *Pour tout graphe rationnel (resp. synchronisé) G et pour tout couple d'ensembles rationnels I et F, il existe un graphe rationnel (resp. synchronisé) G', un sommet i et un ensemble rationnel F' tels que $L(G,I,F) = L(G',\{i\},F')$.*

Démonstration. Soit $G = (T_a)_{a\in\Sigma}$ un graphe rationnel à sommets dans Γ^* et soient i un symbole n'appartenant pas à Γ et $\Gamma' = \Gamma \cup \{i\}$. Pour tout $a \in \Sigma$, soit T'_a un transducteur reconnaissant la relation rationnelle $T_a \cup \{(i,w) \mid w \in T_a(I)\}$. Remarquons que si T_a est synchronisé alors T'_a peut également être choisi synchronisé. Si $\varepsilon \notin L(G,I,F)$ alors $F' = F$, sinon $F' = F \cup \{i\}$. Il n'est pas difficile de montrer que $L(G,I,F) = L(G',\{i\},F')$. □

Il découle de la proposition 4.11 et du lemme 4.23 que les graphes rationnels synchronisés à sommet initial unique acceptent tous les langages contextuels [Ris02].

Remarque 4.24. Il est assez évident que ce résultat n'est plus vrai dans le cas des graphes synchrones : en effet, la restriction d'un graphe synchrone à l'ensemble des sommets accessibles depuis un ensemble fini de sommets initiaux est fini. Donc le langage d'un graphe synchrone depuis un ensemble fini de sommets est un langage rationnel. De façon similaire, comme tout langage rationnel est accepté par un graphe fini, il peut aussi être accepté par un graphe séquentiel synchrone à un seul sommet initial.

Remarquons que la construction du lemme 4.23 utilise le degré infini pour transformer un graphe synchrone avec un ensemble rationnel de sommets en un graphe rationnel équivalent à sommet initial unique. Afin d'obtenir des notions plus satisfaisantes d'automates infinis et de leurs langages, nous restreignons maintenant notre attention aux graphes de degré fini à sommet initial unique.

4.2.3.1 Graphes rationnels : degré fini et sommet initial unique

Nous présentons une transformation syntaxique des graphes rationnels synchrones à ensemble rationnel de sommets initiaux en graphes rationnels de degré fini à sommet initial unique acceptant le même langage.

Cette construction repose sur le fait que pour qu'un graphe synchrone reconnaisse un mot de longueur $n > 0$, il est seulement nécessaire de considérer les sommets de taille inférieure à c^n (où c est une constante dépendant uniquement

du graphe). Nous commençons par établir un résultat similaire pour les systèmes de pavage et concluons en utilisant la correspondance étroite entre graphes synchrones et systèmes de pavage établie à la proposition 4.15.

Lemme 4.25. *Pour tout système de pavage $S = (\Gamma,\Sigma,\#,\Delta)$, si $p \in P(S)$ alors il existe une image p' de dimensions (n,m) telle que* $\mathrm{fr}\,(p) = \mathrm{fr}\,(p')$ *et $n \leq |\Gamma|^m$.*

Démonstration. Soit p' une image de dimensions (n,m) avec $n > |\Gamma|^m$, et supposons que p' est la plus petite image dans $P(S)$ dont la frontière est $\mathrm{fr}\,(p)$. Soient $l_1,\ldots,l_n$ les lignes successives de p'. Comme $n > |\Gamma|^m$, il existe $j > i \geq 1$ tels que $l_i = l_j$. Soit p'' l'image de lignes $l_1,\ldots,l_i,l_{j+1},\ldots,l_n$. Il est facile de vérifier que $T({p''}_{\#}) \subset T({p'}_{\#})$. Nous avons donc $p'' \in P(S)$ et comme p'' a une hauteur plus petite que p' mais la même frontière, on obtient une contradiction. □

Nous savons d'après la remarque 4.16 que pour tout graphe synchrone $G = (T_a)_{a\in\Sigma}$ et tout couple d'ensembles rationnels I et F, il existe un système de pavage S tel que $i \xrightarrow[G]{w} f$ avec $i \in I$ et $f \in F$ si et seulement si il existe $p \in P(S)$ telle que $\mathrm{fr}\,(p) = w$ et p a pour hauteur $|i| = |f|$. Donc, comme conséquence directe du lemme 4.25, on obtient la propriété suivante.

Lemme 4.26. *Pour tout graphe rationnel synchrone G et ensembles rationnels I et F, il existe $k \geq 1$ tel que*

$$\forall w \in L(G,I,F),\exists i \in I,f \in F \text{ tel que } i \xrightarrow[G]{w} f \text{ et } |i| = |f| \leq k^{|w|}.$$

Nous pouvons maintenant présenter la construction d'un graphe rationnel de degré fini acceptant depuis un unique état initial le même langage que tout graphe synchrone d'ensemble de sommets initiaux rationnel.

Proposition 4.27. *Pour tout graphe rationnel synchrone G et tous ensembles rationnels I et F tels que $I \cap F = \emptyset$, il existe un graphe rationnel H de degré fini et un sommet i tels que $L(G,I,F) = L(H,\{i\},F)$.*

Démonstration. D'après le lemme 4.14, il existe un graphe rationnel synchrone R décrit par un ensemble de transducteurs $(T_a)_{a\in\Sigma}$ sur Γ^* tel que $L(G,I,F) = L(R,\#^*,F)$. Notons que pour tous $w \in \#^*$ et $w' \in \Gamma^*$, si $w \xrightarrow[R]{} w'$ alors w' ne contient pas $\#$. On définit un graphe rationnel H tel que $L(G,I,F) = L(H,\{i\},F)$ pour un certain sommet i de H. Soient k la constante évoquée au lemme 4.26 et T et T' deux transducteurs réalisant les relations rationnelles $\{(\#^n,\#^{kn}) \mid n \in \mathbb{N}\}$ et $\{\#^n,\#^m \mid m \in [1,n]\}$ respectivement. Pour tous $a,b,c \in \Sigma$ et $u \in \Sigma^*$, H possède

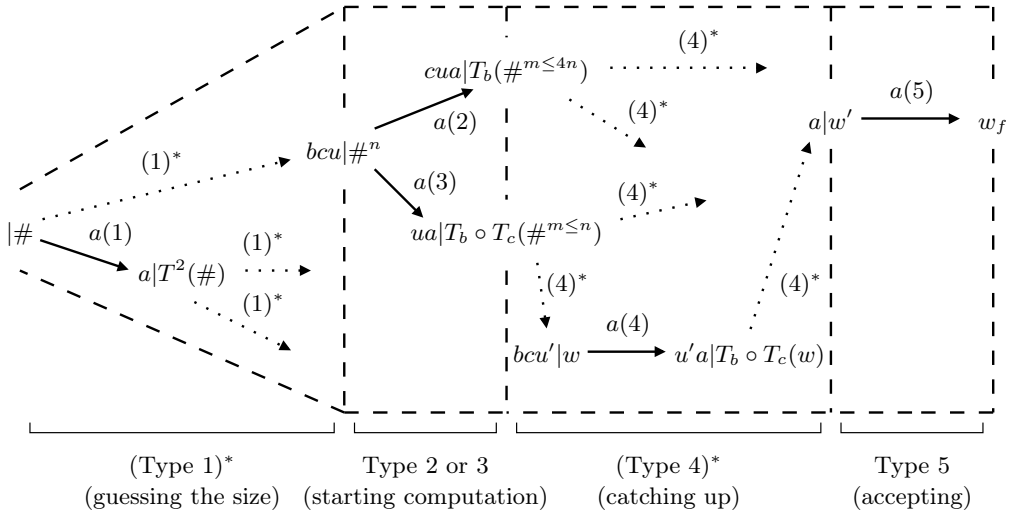


FIG. 4.10 – *Schéma de la construction de la proposition 4.27.*

---

les arcs suivants :

$$\begin{array}{rrcll}
\forall n \in \mathbb{N}, & u|\#^n & \xrightarrow{a} & ua|\, T \circ T(\#^n) & \text{(Type 1)} \\
\forall n \in \mathbb{N}, & bu|\#^n & \xrightarrow{a} & ua|\, T \circ T' \circ T_b(\#^n) & \text{(Type 2)} \\
\forall n \in \mathbb{N}, & bcu|\#^n & \xrightarrow{a} & ua|\, T' \circ T_b \circ T_c(\#^n) & \text{(Type 3)} \\
\forall w \in (\Gamma \setminus \{\#\})^*, & bcu|w & \xrightarrow{a} & ua|\, T_b \circ T_c(w) & \text{(Type 4)} \\
\forall w \in (\Gamma \setminus \{\#\})^*, & b|w & \xrightarrow{a} & T_b \circ T_a(w) & \text{(Type 5)} \\
& |\# & \xrightarrow{a} & T \circ T' \circ T_a(\#) & \text{(Type 6)}
\end{array}$$

Le graphe H est clairement rationnel et de degré fini. Nous prenons $i = |\#$ comme sommet initial et prétendons que $L(R,\#^*,F) = L(H,\{i\},F)$.

Remarquons que dans H un arc de type 2 ou 3 ne peut être suivi d'arcs de type 1, 2 ou 3, et au plus un arc de type 2 ou 3 et de type 5 ou 6 peut être appliqué dans un même chemin. De plus, un arc de type 1 augmente la taille de la partie gauche d'un mot de 1, et un arc de type 4 la diminue de 1. Enfin, dans tout chemin acceptant, le dernier arc doit être de type 5 ou 6. La figure 4.10 illustre la structure du graphe obtenu.

Montrons tout d'abord que pour tout chemin acceptant $i = c_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} c_n \in F$ pour $w = a_1 \dots a_n$ dans H, il existe un chemin acceptant $c'_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} c'_n = c_n \in F$ pour w dans R. Nous distinguons trois cas :

1. Cas $n = 1$: l'unique arc doit être de type 6. Donc $T_{a_1}(T'(T(\#))) \cap F$ n'est pas vide. On peut choisir $c'_0 = T'(T(\#))$ et $c'_1 = T_{a_1}(c'_0)$, donc $w = a_1$ appartient à $L(R,\#^*,F)$.

2. Cas $n = 2m$ avec $m > 0$: le chemin de H est composé de m arcs de type 1 suivi d'un arc de type 3, $m-2$ arcs de type 4 et enfin un arc de type 5. Nous avons donc :
   (a) pour tout $i \in [1,m]$, $c_i = a_1 \dots a_i | T^{2i}(\#)$,
   (b) pour tout $i \in [1,m-1]$ on a $c_{i+m} = a_{2i+1} \dots a_{m+i} | c'_{2i}$ avec $c'_0 \in T^{2m} \circ T'(\#)$ et $c'_{2i} \in T_{a_1} \circ \dots \circ T_{a_{2i}}(c'_0))$,
   (c) $c_n = c'_n \in T_{a_1} \circ \dots \circ T_{a_n}(c'_0)$.

   Nous avons déjà fixé c'_k pour tout indice pair k entre 0 et n. Pour tout $i \in [1,m]$, on peut choisir pour c'_{2i-1} tout mot de $T_{a_{2i-1}}(c'_{2i-2})$ tel que $c'_{2i} = T_{a_{2i}}(c'_{2i-1})$ (qui existe forcément par construction de c'_{2i-2} et c'_{2i}). Comme c'_0 est dans $\#^*$ et $c'_n = c_n \in F$, cela signifie que $w \in L(R,\#^*,F)$.
3. Cas $n = 2m+1$ avec $m > 0$: le chemin est composé de m arcs de type 1 suivis d'un arc de type 2, $m-1$ arcs de type 4 et enfin un arc de type 5. Nous avons donc :
   (a) pour tout $i \in [1,m]$, $c_i = a_1 \dots a_i | T^{2i}(\#)$,
   (b) soit $c'_0 \in T^{2m+1} \circ T'(\#)$, on a $c_{m+1} = a_2 \dots a_{m+1} | c'_1$ pour un certain $c'_1 \in T_{a_1}(c'_0)$,
   (c) pour tout $i \in [1,m-1]$ on a $c_{i+m+1} = a_{2i+2} \dots a_{m+i} | c'_{2i+1}$ avec $c'_{2i+1} \in T_{a_1} \circ \dots \circ T_{a_{2i+1}}(c'_0)$,
   (d) $c_n = c'_n \in T_{a_1} \circ \dots \circ T_{a_n}(c'_0)$.

   Nous avons déjà fixé c'_0 ainsi que tous les c'_k pour les indices impairs k entre 0 et n. Pour tout $i \in [1,m]$, on peut choisir pour c'_{2i} tout mot dans $T_{a_{2i}}(c'_{2i-1})$ tel que $c'_{2i+1} = T_{a_{2i+1}}(c'_{2i})$ (qui existe nécessairement par construction de c'_{2i-1} et c'_{2i+1}). Comme c'_0 appartient à $\#^*$ et $c'_n = c_n \in F$, on a bien $w \in L(R,\#^*,F)$.

Nous avons démontré que $L(H,\{i\},F) \subseteq L(R,\#^*,F)$.

Montrons à présent la réciproque, c'est à dire que pour tout chemin acceptant $c_0 \in \#^* \xrightarrow{a_1} \dots \xrightarrow{a_n} c_n \in F$ dans R avec $w = a_1 \dots a_n$, il existe un chemin acceptant pour w dans H. D'après le lemme 4.26 et par définition de k, on peut supposer que $|c_0| = |c_n| \leq k^n$. Il n'est pas difficile de montrer que pour tout mot x dans $T_{a_1} \circ \dots \circ T_{a_n}(T^n \circ T'(\#))$, $i \xrightarrow[H]{w} x$. Il reste à montrer que c_n appartient bien à $T_{a_1} \circ \dots \circ T_{a_n}(T^n \circ T'(\#))$. Par définition de T et T', $T^n \circ T'(\#)$ est égal à $\{\#^i \mid i \in [1,k^n]\}$. Donc comme $c_0 \in T^n \circ T'(\#)$, il s'ensuit que $c_n \in T_{a_1} \circ \dots \circ T_{a_n}(T^n \circ T'(\#))$. Par conséquent $i \Longrightarrow_R^w c_n$, et comme $c_n \in F$, w appartient à $L(H,\{i\},F)$. Ceci montre que $L(R,\#^*,F) \subseteq L(H,\{i\},F)$. □

Des propositions 4.11 et 4.27, on déduit que les graphes rationnels de degré fini à sommet initial unique acceptent tous les langages contextuels. Ce résultat

fur déjà obtenu dans [MS01] en utilisant la forme normale de Penttonen des grammaires contextuelles.

Théoreme 4.28. *Les langages des graphes rationnels de degré fini à sommet initial unique et ensemble rationnel de sommets finaux sont les langages contextuels.*

4.2.3.2 Graphes synchronisés : degré fini et sommet initial unique

Nous considérons maintenant les langages des graphes synchronisés de degré fini à sommet initial unique. Tout d'abord, nous les caractérisons comme les langages acceptés par les systèmes de pavage à images carrées (c'est à dire pour lesquels il existe $c \in \mathbb{N}$ tel que pour tout mot $w \in L(S)$, il existe une image de dimensions (n,m) dans $P(S)$ telle que $n \leq cm$ de frontière w). Une légère adaptation de la construction de la proposition 4.27 nous donne la première inclusion.

Proposition 4.29. *Soit $S = (\Gamma,\Sigma,\#,\Delta)$ un système de pavage à images carrées. Il existe un graphe rationnel synchronisé de degré fini acceptant $L(S)$ depuis un sommet initial unique.*

Démonstration. Soit $G = (T_a)_{a\in\Sigma}$ le graphe synchronisé obtenu à partir de S suivant la construction de la proposition 4.11. Dans la construction de la proposition 4.27, si l'on remplace le transducteur T par un transducteur R réalisant la relation synchronisée $\{(\#^n,\#^{n+c}) \mid n \in \mathbb{N}\}$, on obtient un graphe synchronisé de degré fini H, un sommet initial i et un ensemble de sommets finaux F tels que $L(H,i,F) = L(S)$. □

Avant de passer à la réciproque, nous établissons un résultat semblable au lemme 4.26 pour les graphes synchronisés *de degré fini* qui indique que lorsqu'on reconnaît un mot w *depuis un unique sommet initial* i, les sommets impliqués ont une longueur au plus linéaire en la taille de w.

Lemme 4.30. *Pour tout graphe rationnel synchronisé de degré fini G à sommets dans Γ^* et pour tout sommet i, il existe une constante k telle que pour tout w dans $L(G,\{i\},F)$, il existe un chemin de i à un certain $f \in F$, étiqueté par w, dont les sommets ont une taille d'au plus $k \cdot |w|$.*

Démonstration. Il découle de la définition des transducteurs synchronisés que pour tout transducteur synchronisé de degré fini il existe $c \in \mathbb{N}$ tel que $(x,y) \in T$ implique que $|x| \leq |y| + c$ (voir [Sak03] pour une démonstration de ce résultat). On choisit k comme le maximum de ces valeurs sur l'ensemble des transducteurs définissant G. Le résultat s'obtient par un raisonnement par récurrence simple sur la longueur de w. □

L'inclusion inverse est obtenue en remarquant que la composition des constructions des propositions 4.10 et 4.15 produit un système de pavage à images carrées lorsqu'elle est appliquée à un graphe synchronisé de degré fini.

Proposition 4.31. *Soit $G = (T_a)_{a\in\Sigma}$ un graphe synchronisé de degré fini. Pour tout sommet i de G et tout ensemble F de sommets finaux, il existe un système de pavage S à images carrées tel que $L(S) = L(G,\{i\},F)$.*

Démonstration. Soient G', I' et F' le graphe synchrone et les ensembles rationnels de sommets initiaux et finaux obtenus en appliquant la construction de la proposition 4.10 à G, $\{i\}$ et F. On peut montrer facilement que pour tout $w \in L(G',I',F')$, il existe $i' \in I'$ et $f' \in F'$ tels que $i' \xrightarrow{w} f'$ avec $|i'| = |f'| \leq k|w|$ où k est la constante su lemme 4.30 pour G. Nous concluons par la proposition 4.15, qui assure l'existence d'un système de pavage S tel que $L(S) = L(G',I',F')$. D'après la remarque 4.13, S est un système de pavage à image carrées. □

En combinant les propositions 4.29 et 4.31 et à l'aide du résultat de simulation du théorème 4.6, on obtient le théorème suivant.

Théoreme 4.32. *Les langages acceptés par les graphes synchronisés de degré fini depuis un unique sommet initial vers un ensemble rationnel de sommets finaux sont les langages contextuels acceptés par des machines linéairement bornées en un nombre linéaire de nombre de renversements de tête de lecture.*

Nous conjecturons que cette classe est strictement contenue dans l'ensemble des langages contextuels. Cependant, peu de résultats de séparation existent pour les classes de complexité restreintes à la fois en temps et en espace (voir par exemple [vM04]). En particulier, les techniques habituelles de diagonalisation (voir [For00]) utilisées pour démontrer que la hiérarchie polynomiale en temps (sans contrainte d'espace) est stricte ne s'appliquent pas par manque d'une notion adaptée de LBM *universelle*.

4.2.3.3 Borner le degré

Dans la continuité de nos restrictions progressives de la famille de graphes considérée, il est naturel de se demander si les graphes rationnels acceptent toujours l'ensemble des langages contextuels si l'on borne leur degré sortant. Cette question est difficile, et nous n'y donnons qu'une réponse très partielle dans le cas des graphes synchronisés de degré borné.

Il découle du lemme 4.30 que les sommets utilisés dans un graphe rationnel synchronisé pour reconnaître un mot w ont une taille au plus linéaire en la longueur de w, et de ce fait peuvent être stockés sur la bande de travail d'une LBM. De plus, si le graphe est déterministe, on peut utiliser une LBM déterministe pour reconnaître son langage.

Proposition 4.33. *Le langage accepté par un graphe rationnel synchronisé déterministe depuis un sommet initial unique est contextuel déterministe.*

Démonstration. Soit $G = (T_a)_{a\in\Sigma}$ un graphe rationnel synchronisé déterministe sur Γ, i un sommet et F un ensemble rationnel de sommets. Nous définissons une LBM déterministe M acceptant $L(G,\{i\},F)$. Pour accepter un mot $w = a_1 \dots a_{|w|}$, M commence par écrire i sur sa bande. Elle applique ensuite successivement T_{a_1}, $\dots$, $T_{a_{n-1}}$ et T_{a_n} à i. Si l'image de la bande courante par l'un de ces transducteurs n'est pas définie, la machine rejette le mot. Sinon, elle vérifie si le dernier contenu de bande obtenu représente un sommet de F.

Nous détaillons à présent comment la machine M peut appliquer de façon déterministe à un mot x l'un des transducteurs T de G. Comme T a une image finie pour tout mot d'entrée, on peut supposer sans perte de généralité que $T = (\Gamma,Q,i,F,\delta)$ est en forme quasi real-time : $\delta \subset Q \times \Gamma \times \Gamma^* \times Q$ (voir par exemple [Ber79] pour une présentation de ce résultat). La machine énumère tous les calculs de T de longueur inférieure à $c|x|$ dans l'ordre lexicographique, où c est la constante associée à G dans le lemme 4.30. Pour tout tel chemin ρ, elle vérifie s'il s'agit d'un chemin acceptant pour l'entrée x, et dans ce cas remplace x par la sortie de ρ.

L'espace utilisé par M quand elle démarre sur un mot w est borné par $(2c+1)|w|$. De plus, si M accepte w, alors il existe un chemin étiqueté par w de i à un sommet $f \in F$ dans G. Réciproquement, si w appartient à $L(G,\{i\},F)$ alors d'après le lemme 4.30, il existe un chemin dans G de i à F ne contenant que des sommets de longueur inférieure à $c|w|$, et par construction M accepte w. Donc, M est une machine linéairement bornée déterministe acceptant $L(G,\{i\},F)$. □

Remarque 4.34. Le résultat de la proposition 4.33 s'étend à tout graphe rationnel déterministe satisfaisant la propriété exprimée par le lemme 4.30.

Le résultat précédent peut être étendu aux graphes synchronisés de degré sortant borné grâce à un résultat d'uniformisation dû à Weber. Observons tout d'abord qu'un graphe rationnel est de degré sortant borné par une certaine constante si et seulement si il est défini par des transducteurs qui associent chacun au plus k images distinctes à chaque mot d'entrée. Les relations réalisées par ces transducteurs sont appelées relations rationnelles k-valuées.

Proposition 4.35 ([Web96])**.** *Pour toute relation rationnelle k-valuée R, il existe k relations rationnelles fonctionnelles $F_1,\dots,F_k$ telles que $R = \bigcup_{i\in[1,k]} F_i$.*

Notons que même si R est une relation synchronisée, les relations F_i ne le sont pas forcément. Cependant, elles vérifient toujours l'inégalité $|y| \leq |x| + c$ pour tout $(x,y) \in F_i$.

À tout graphe synchronisé G de degré sortant borné par k défini par un ensemble de transducteurs $(T_a)_{a\in\Sigma}$, on associe le graphe rationnel déterministe H défini par $(F_{a_i})_{a\in\Sigma,i\in[1,k]}$ où pour tout $a \in \Sigma$, $(F_{a_i})_{i\in[1,k]}$ est l'ensemble des fonctions rationnelles associées à T_a par la proposition 4.35. D'après la proposition 4.33 et la remarque 4.34, $L(H,\{i\},F)$ est un langage contextuel déterministe. Soit

π la projection alphabétique définie par $\pi(a_i) = a$ pour tout $a \in \Sigma$ et $i \in [1,k]$, il n'est pas difficile d'établir que $\pi\left(L(H,\{i\},F)\right) = L(G,\{i\},F)$. Comme les langages contextuels déterministes sont clos par projection alphabétique, $L(G,\{i\},F)$ est un langage contextuel déterministe.

Théoreme 4.36. *Le langage accepté par un graphe synchronisé de degré borné depuis un sommet initial unique est contextuel déterministe.*

La réciproque à ce résultat n'est pas claire, pour des raisons semblables à celles présentées dans le paragraphe précédent sur les graphes synchronisés de degré fini. Il serait intéressant de parvenir à donner une caractérisation précise des langages des graphes rationnels synchronisés de degré borné.

4.2.4 Notions de déterminisme

Dans cette dernière partie du travail consacré aux graphes rationnels, nous nous penchons sur des familles de graphes acceptant les langages contextuels déterministes. Tout d'abord, nous examinons la famille de graphes obtenue en appliquant les constructions précédentes à des langages déterministes. Puis, nous proposons une condition globale sur les ensembles de transducteurs telle que la sous-famille de graphes rationnels obtenus accepte exactement l'ensemble des langages contextuels déterministes.

4.2.4.1 Langages contextuels non-ambigus

Dans la démonstration de la proposition 4.11, étant donné un système de pavage quelconque nous décrivons un moyen de construire un graphe rationnel synchrone acceptant le même langage. Une question naturelle est donc de déterminer la famille de graphes obtenus lorsque cette construction est appliquée exclusivement dans le cas déterministe. Lorsqu'on applique cette construction à un système de pavage déterministe, on obtient un graphe rationnel synchrone G, qui est non-déterministe en général, et deux ensembles rationnels I et F de sommets tels que $L(G,I,F) = L(S)$, avec la particularité que pour tout mot w dans L il existe *un et un seul* chemin étiqueté par w menant d'un sommet de I à un sommet de F. G est *non-ambigu* par rapport à I et F.

Cependant, la réciproque n'est pas assurée : étant donné un graphe G et deux ensembles rationnels I et F tels que G est non-ambigu par rapport à I et F, on ne peut pas directement assurer que $L(G,I,F)$ est contextuel déterministe. En effet, si nous appliquons la construction de la proposition 4.15, on obtient un système de pavage tel que toute paire d'images distinctes acceptée possède des frontières différentes. Remarquons qu'un tel système de pavage acceptant au plus une image p avec une frontière w donnée pour tout $w \in L(S)$ n'est pas nécessairement déterministe. En fait, on peut facilement s'apercevoir qu'ils correspondent

aux machines linéairement bornées non-ambigus (c'est à dire les machines déterministes ayant au plus un calcul acceptant par mot). La famille de langages de ces machines est appelée USPACE(n), et on ne sait pas à l'heure actuelle si cette classe coïncide soit avec les langages non-déterministes soit avec les contextuels déterministes. Plutôt que le résultat espéré sur les langages déterministes, cette approche nous permet donc seulement d'obtenir le résultat intermédiaire suivant.

Théoreme 4.37. *Soit L un langage, les propriétés suivantes sont équivalentes :*

1. *L est un langage contextuel non-ambigu.*
2. *Il existe un graphe rationnel G à transducteurs non-ambigus et deux ensembles rationnels I et F tels que $L = L(G,I,F)$ et G est non-ambigu par rapport à I et F.*

Ce résultat n'est vrai que si l'on considère des transducteurs non-ambigus, c'est à dire des transducteurs dans lesquels il existe au plus un chemin acceptant par couple de mots. La raison pour ceci est que l'ambiguïté d'un transducteur introduirait une ambiguïté dans le système de pavage correspondant. Cependant, nous savons que dans le cas des transducteurs synchronisés il est possible de rendre tout transducteur non-ambigu. (cf. remarque 1.28). On obtient alors le corollaire suivant.

Corollaire 4.38. *Les langages des graphes synchronisés non-ambigus entre des ensembles rationnels de sommets initiaux et finaux sont les langages contextuels non-ambigus.*

Notons que la non-ambiguïté d'un graphe rationnel ou synchronisé est une propriété indécidable, et que donc ces classes de graphes acceptant les langages non-ambigus ne sont pas récursives. Cependant, on peut faire l'observation suivante : comme toute fonction rationnelle peut être réalisée par un transducteur non-ambigu [Kob69, Sak03], les langages des graphes rationnels déterministes sont, d'après le théorème 4.37, des langages contextuels non-ambigus.

Corollaire 4.39. *Le langage accepté par un graphe rationnel déterministe depuis un sommet initial unique vers un ensemble rationnel de sommets finaux est un langage contextuel non-ambigu.*

Démonstration. Soit $G = (T_a)_{a\in\Sigma}$ un graphe rationnel déterministe. Pour tout $a \in \Sigma$, la relation F_a reconnue par T_a est une fonction, et suivant [Kob69, Sak03] il existe un transducteur non-ambigu T'_a la reconnaissant. D'après le théorème 4.37, on sait donc que pour tout $i \in I$, $L(G,\{i\},F)$ est un langage contextuel non-ambigu. □

4.2.4.2 Ensembles de transducteurs globalement déterministes

Nous venons de voir une tentative de caractériser des familles naturelles de graphes dont les langages sont les langages contextuels déterministes, fondée sur

la restriction des constructions précédentes au cas déterministe, mais qui n'a pas atteint son objectif à cause d'une légère nuance entre les notions de déterminisme et de non-ambiguïté pour les systèmes de pavage.

Premièrement nous considérons la famille naturelle des graphes séquentiels synchrones à ensemble de sommets initiaux de la forme i^*. Il est facile de vérifier que l'application de la construction de la proposition 4.15 à l'un de ces graphes produit un système de pavage déterministe.

Proposition 4.40. *Le langage d'un graphes séquentiel synchrone depuis un ensemble i^* de sommets est contextuel déterministe.*

La réciproque semble difficile à démontrer en raison de la nature locale du déterminisme mis en jeu dans ce cas. Par conséquent, nous considérons une propriété *globale* de l'ensemble des transducteurs caractérisant un graphe rationnel de telle sorte que chaque chemin acceptant corresponde à un calcul acceptant de machine linéairement bornée déterministe sur l'entrée correspondante, ou de façon équivalente à une image acceptée par un système de pavage déterministe et dont la frontière supérieure est l'étiquette du chemin considéré.

Pour tout langage rationnel L, nous notons T_L le transducteur synchrone minimal reconnaissant la relation identité sur L.

Définition 4.41. Soit T un ensemble de transducteurs synchrones sur Γ. On dit de T qu'il est *globalement déterministe* par rapport à deux langages I et $F \subseteq \Gamma^*$ si tout transducteur dans T est déterministe[4] et si pour chaque paire de transducteurs $T_1 \in T \cup \{T_I\}$ et $T_2 \in T \cup \{T_F\}$, et toute paire d'états de contrôles $q_1 \in Q_{T_1}$ et $q_2 \in Q_{T_2}$, il existe au plus un b tel que

$$q_1 \xrightarrow[T_1]{a/b} q_1' \wedge q_2 \xrightarrow[T_2]{b/c} q_2' \text{ avec } a,c \in \Gamma,\ q_1' \in Q_{T_1},\ q_2' \in Q_{T_2}.$$

Intuitivement, cette condition signifie qu'à chaque fois qu'une partie de la sortie d'un transducteur peut être lue par un second transducteur, il existe un plus une façon de rajouter une lettre à ce mot de façon à ce qu'il reste compatible avec les deux transducteurs. Cette propriété est trivialement décidable, puisqu'il suffit de vérifier la condition ci-dessus pour chaque paire d'états de contrôle des transducteurs dans $(T \cup \{T_I\}) \times (T \cup \{T_F\})$. Ceci nous permet de capturer une sous-famille des graphes rationnels dont les langages sont précisément les langages contextuels déterministes.

Théoreme 4.42. *Soit L un langage, les deux propriétés suivantes sont équivalentes :*

1. *L est un langage contextuel déterministe.*
2. *Il existe un graphe rationnel synchrone G et deux ensembles rationnels I et F tels que $L = L(G,I,F)$ et G est globalement déterministe entre I et F.*

4. C'est à dire qu'à chaque fois que $q \xrightarrow{a/b} q'$ et $q \xrightarrow{c/d} q''$ avec $q' \neq q''$, on a $(a,b) \neq (c,d)$.

Démonstration. Soit $G = (T_a)_{a\in\Sigma}$ un graphe rationnel synchrone globalement déterministe entre I et F. Le graphe $H = (T'_a)_{a\in\Sigma}$ obtenu en appliquant le lemme 4.14 à G est tel que $L(H,i^*,f^*) = L(G,I,F)$. De plus, H est globalement déterministe entre i^* et f^*.

Nous allons montrer que la construction de la proposition 4.15 appliquée au graphe H engendre un système de pavage déterministe. Supposons que ce ne soit pas le cas. Alors, par définition d'un système de pavage non-déterministe, il doit exister des mots u, v_1 et v_2 avec $v_1 \neq v_2$ tels que les images à deux lignes p_1 et p_2 dont la première ligne est $\#u\#$ et la seconde respectivement $\#v_1\#$ et $\#v_2\#$ n'ont que des tuiles dans Δ. Comme $v_1 \neq v_2$, soit i le plus petit indice tel que $v_1(i) \neq v_2(i)$. Soit $v_1(i) = xp$, $v_2(i) = x'p'$.

Selon la construction de la proposition 4.15, il existe deux transducteurs T_a et T_b tels que

$$q_a \xrightarrow[T_a]{y/x} p \wedge q_b \xrightarrow[T_b]{x/z} q'_b \wedge q_a \xrightarrow[T_a]{y/x'} p' \wedge q_b \xrightarrow[T_b]{x'/z'} q''_b$$

pour des symboles $y,y',z,z' \in \Gamma$ et des états de contrôle q_a,q_b,q'_b et q''_b. Comme T_a est déterministe, si x est égal à x', alors $p = p'$ et $v_1(i) = v_2(i)$. Donc $x \neq x'$, et les relations ci-dessus contredisent le déterminisme global de H.

Réciproquement, soit L un langage contextuel déterministe. D'après le théorème 4.7, il existe un automate cellulaire déterministe $C = (\Gamma,\Sigma,\perp,\delta,[,])$ acceptant L. Nous allons construire deux langages rationnels I et F et un ensemble de transducteurs T globalement déterministe par rapport à I et F tels que $L(G,I,F) = L$ où G est le graphe rationnel défini par T. L'alphabet de travail de T est $\Gamma' = \Sigma \cup \{[,]\} \cup \delta$. L'ensemble d'états de contrôle de chaque transducteur $T_a \in T$ est $\{q^a_0\} \cup \{q^a_{AB} \mid A,B \in \Gamma \cup \{[]\}\}$, où q^a_0 est l'unique état initial. Ses transitions sont

$$\forall a,b \in \Sigma, \quad q^a_0 \xrightarrow{[/a} q^a_{[a} \quad \text{et} \quad q^a_0 \xrightarrow{b/a} q^a_{ba}$$

$$\forall d_1 = ([,A,B,A') \in \delta, \quad q^a_{[A} \xrightarrow{[/d_1} q^a_{[A'}$$

$$\forall d_1 = (A,B,C,B'),d_2 = (B,C,D,C') \in \delta, \quad q^a_{BC} \xrightarrow{d_1/d_2} q^a_{B'C'}$$

Les états finaux de T_a sont $q_{[\perp}$ et $q_{\perp\perp}$. Soit maintenant $I = ([)^*$ et $F = \Sigma R^*$ avec $R = \{(a,b,],b') \in \delta \mid a,b,b' \in \Gamma\}$. Par construction et comme C est déterministe, T est globalement déterministe par rapport à I et F. Vérifions que $L(G,I,F) = L$.

$L(G,I,F) \subseteq L$: soit $v_0,\ldots,v_n$ un chemin acceptant étiqueté par w dans G, avec $v_0 = ([)^{m+1} \in I$, $v_n \in F$, $n = |w|$ et $v_i(1) = w(i)$ pour tout $i > 0$. On appelle p l'image de dimensions $m \times n$ dont les colonnes sont $v_0,\ldots,v_n,v_{n+1}$ (avec $v_{n+1} = (])^{m+1}$), et $u_0 \ldots u_m$ les lignes de p. Notons π le morphisme sur $(\Sigma \cup \{[,]\} \cup \delta)^*$ défini par $\pi(a) = a$ pour tout $a \in \Sigma$, $\pi([) = [$, $\pi(]) =]$ et $\pi(d) = B$ pour tout $d = (A,B,C,D) \in \delta$. Par construction de T, pour tout $i \in [2,m]$ et

$j \in [1,n]$, on a $\pi(u_i(j)) = B'$ si et seulement si il existe des lettres A,B et $C \in (\Gamma \cup \{[,]\})$ telles que $u_{i-1}(j) = (A,B,C,B') \in \delta$, ou en d'autres termes :

$$\pi(u_{i-1}(j-1)) = A,\ \pi(u_{i-1}(j)) = B,\ \pi(u_{i-1}(j+1)) = C \ \wedge \begin{array}{|c|c|c|}\hline A & B & C \\ \hline \multicolumn{1}{c}{} & B' & \multicolumn{1}{c}{} \\ \cline{2-2}\end{array} \in \delta.$$

Ceci implique que pour tout $i \in [2,m]$, il existe une transition dans C entre les configurations représentées par u_{i-1} et u_i selon δ, ou en d'autres termes $(u_1,\ldots,u_m)$ est un calcul valide de C entre u_1 et u_m. Comme de plus $\pi(u_1) = [w]$, et comme par définition des états finaux de T il doit exister une transition dans C entre $\pi(u_m)$ et $[\perp^n]$, w est accepté par C.

$L \subseteq L(G,I,F)$: soit $w \in L$, et soit $(u_1 = [w],u_2,\ldots,u_m = [\perp^n])$, avec $n = |w|$, un calcul acceptant de C sur w. Soit u'_i, $i \in [1,m-1]$ le mot sur δ^* définit comme suit :

$$u'_i(1) = [\,,\quad u'_i(n+1) =]\ \text{ et }\ \forall j \in [2,n+1],\ u'_i(j) = d_{i,j} \text{ avec } \big(u_i(j-1),u_i(j),u_i(j+1),u_{i+1}(j)\big) \in \delta.$$

En d'autres termes, $u'i = [d_1 \ldots d_n]$, où chaque d_j est l'unique règle de δ utilisée à l'indice j pour transformer u_i en u_{i+1}. Considérons maintenant l'image p formée des lignes u'_1 à u'_{m-1}, et soient $v_1,\ldots,v_{n+2}$ les colonnes de p. Il est assez direct de montrer d'après la construction de T que $([v_1,w(1)v_2) \in T_{w(1)}$ et $(w(j)v_{j+1},w(j+1)v_{j+2}) \in T_{w(j+1)}$ pour tout $j \in [1,n-1]$. On a aussi $[v_1 \in I$ et $w(n)v_{n+1} \in F$, et donc w étiquette un chemin acceptant dans G. □

Exemple 4.43. La figure 4.11 illustre la construction précédente appliquée à l'automate cellulaire déterministe de l'exemple 4.5 acceptant le langage $\{a^n b^n \mid n \geq 1\}$. Les transitions des transducteurs sont étiquetées soit par des lettres de $\Sigma \cup \{[\}$ soit par des transitions d'automate cellulaire désignées par leur numéro (voir la figure 4.4).

4.3 Graphes linéairement bornés

Parmi toutes les méthodes présentées au paragraphe 2.1 pour donner des représentations finies de familles de graphes infinis, et en plus d'autres caractérisations, les graphes préfixe-reconnaissables ainsi que les graphes de machines de Turing possèdent des définitions à la fois comme graphes de transition et comme graphes de type Cayley de systèmes de réécriture. Les graphes rationnels, cependant, n'ont pas de caractérisation comme graphes de transition, ce qui rend la correspondance avec leurs langages moins claire que dans les autres cas, comme

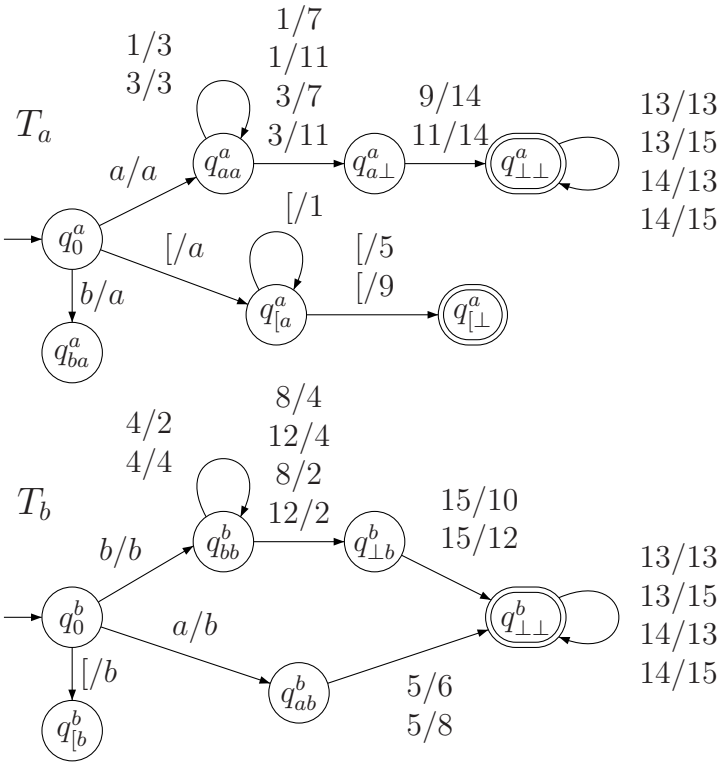


FIG. 4.11 – *Transducteurs d'un graphe synchrone globalement déterministe acceptant le langage* $\{a^n b^n \mid n \geq 1\}$.

nous venons de le voir. Dans ce paragraphe, nous nous intéressons à la définition d'une famille adaptée de graphes de transition de machines linéairement bornées dont les langages soient naturellement les langages contextuels, et à l'étude des propriétés structurelles de ces graphes.

Nous considérons les graphes de transition des machines linéairement bornées selon la définition du graphe de transition d'une machine donnée au paragraphe 2.1.1.1. Par commodité, nous appelons ces graphes *graphes linéairement bornés*. Une famille semblable a été étudiée dans [KP99, Pay00], où les graphes des configurations des LBM sont étudiés à bisimulation faible près. Cependant, ces travaux ne donnent pas de définition formelle associant les machines linéairement bornées à des graphes temps-réel (sans ε-transitions) représentant leurs calculs observables.

Pour illustrer la pertinence de notre notion, nous fournissons deux caractérisations alternatives des graphes linéairement bornés. Premièrement, nous montrons qu'ils sont isomorphes aux graphes de type Cayley des systèmes de réécriture décroissants. Deuxièmement, nous représentons directement les relations d'arcs d'un graphe linéairement borné à l'aide d'un certain type de transductions contextuelles. Cette triple caractérisation nous permet de déduire aisément des propriétés structurelles des graphes linéairement bornés, comme leur fermeture par produit synchronisé (déjà connue de [KP99]) et par restriction à un ensemble contextuel de sommets.

4.3.1 Définition

4.3.1.1 Graphes de transition de LBM

Nous définissons les graphes linéairement bornés comme la clôture par isomorphisme des graphes de transition de machines linéairement bornées *normalisées*, comme définis au paragraphe 2.1.1.1.

Remarque 4.44. Par simplicité, nous ne considérons que des machines linéairement bornées qui insèrent une nouvelle cellule à chaque fois qu'un symbole est lu en entrée, et dont toute règle entraîne un déplacement sur la gauche ou sur la droite. Des formes plus relâchées où un effacement ou une réécriture peuvent intervenir pendant une lecture peuvent être autorisées sans conséquence sur les résultats. On peut également autoriser les règles qui laissent la tête de lecture immobile.

Nous montrons tout d'abord que, dans le cas des machines linéairement bornées, imposer la normalisation des machines ne réduit pas l'expressivité du modèle. Nous rappelons qu'une machine linéairement bornée est normalisée si, depuis une configuration quelconque, soit on ne peut effectuer que des ε-transitions ou aucune ε-transition. Remarquons que l'ensemble des configurations externes d'une LBM normalisée est un ensemble rationnel, puisque on peut dire si une configura-

tion est externe simplement en inspectant l'ensemble des règles utilisables depuis cette configuration.

Proposition 4.45. *Toute machine linéairement bornée peut être normalisée sans changer son langage accepté.*

Démonstration. Soit M une LBM, on construit une LBM normalisée M' équivalente à M. Sans perte de généralité, supposons que M ne possède aucun cycle étiqueté par ε contenant une configuration acceptante. Cette propriété est similaire à l'aspect « temps réel » d'une machine (cf. proposition 1.10). On suppose également que l'état initial q_0 de M est externe.

Premièrement, M' possède au moins les états de contrôle et les ε-transitions de M, et le même état initial q_0. Pour toute règle de transition $pB \xrightarrow{a} qAB$ de M avec $p \neq q_0$, nous ajoutons à M' un état de contrôle p' et une règle $p'B \xrightarrow{a} qAB$, ainsi que des ε-transitions permettant de passer de p à p' sans déplacer la tête de lecture. Nous prenons ensuite comme ensemble d'états de contrôle externes de M' l'ensemble de tous ces nouveaux états p', qui par définition ne permettent que des Σ-transitions, ainsi que q_0 et les états de M qui n'autorisent aucune transition. De cette façon, la condition de partitionnement sur les états de contrôle est atteinte. Il reste à s'assurer que tous les états finaux de M' sont externes. Pour réaliser ceci, à chaque fois qu'un état final de M est rencontré par M', l'information est transmise le long de chaque ε-transition en marquant l'état de contrôle. Le marquage est réinitialisé dès qu'une nouvelle transition étiquetée est effectuée. D'après l'hypothèse précédente sur l'absence de cycles acceptants étiquetés par ε, tout calcul atteignant une configuration acceptante pour M atteindra forcément en un nombre fini d'étapes une configuration externe de M'. Il ne reste donc qu'à déclarer tous les états externes marqués comme finaux pour que M' soit normalisée et accepte le même langage que M. □

Ce résultat peut s'étendre à la classe entière des machines de Turing. Dorénavant, sauf précision contraire, nous ne considérerons que des LBM normalisées.

Exemple 4.46. La figure 4.12 montre le graphe de transition de la LBM normalisée M de l'exemple 1.9. Cette machine accepte le langage $\{(a^n b^n)^+ \mid n \geq 1\}$, qui est aussi le langage du graphe entre le sommet $[q_0]$ et l'ensemble de sommets $[b^* q_2 b]$. Pour plus de clarté, seule la partie du graphe accessible depuis la configuration $[q_0]$ est représentée. Nous verrons au paragraphe 4.3.2 que ce sous-graphe est lui aussi un graphe linéairement borné.

4.3.1.2 Définitions alternatives

Ce paragraphe fournit deux définitions alternatives des graphes linéairement bornées. Dans [CK02a], il est montré que plusieurs familles de graphes précédemment citées peuvent être exprimées de façon naturelle sous la forme de graphes de

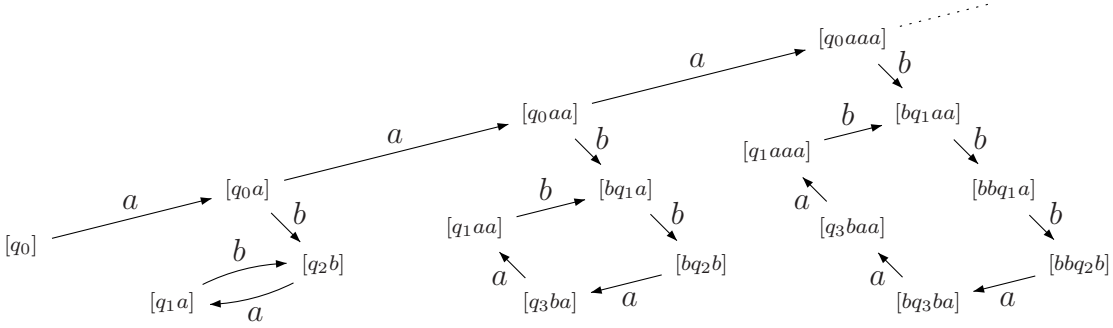


FIG. 4.12 – *Le graphe de transition d'une LBM acceptant* $\{(a^n b^n)^+ \mid n \geq 1\}$.

type Cayley de systèmes de réécritures. Nous montrons que c'est aussi le cas des graphes linéairement bornés, qui sont les graphes de type Cayley des systèmes de réécriture décroissants. La dernière définition de ces graphes que nous présentons change de perspective et définit directement l'ensemble des relations d'arcs d'un graphe linéairement borné à l'aide de transductions contextuelles incrémentales. Cette variété dans les définitions nous permettra par la suite de démontrer plus simplement certaines des propriétés des graphes linéairement bornés.

Graphes de type Cayley de systèmes de réécriture décroissants. Les graphes de type Cayley de systèmes de réécriture sont définis au paragraphe 2.1.1.2. La famille de systèmes à laquelle nous nous intéressons ici est celle des *systèmes de réécriture finis décroissants de mots*, c'est à dire des systèmes de réécriture possédant un ensemble fini de règles de la forme $l \to r$ avec $|l| \leq |r|$, qui peuvent uniquement conserver ou faire décroître la taille des mots auxquels elles s'appliquent. La raison de ce choix est que les relations de dérivation de tels systèmes coïncident avec des compositions arbitraires d'ε-transitions de machines linéairement bornées.

Exemple 4.47. La figure 4.13 montre le graphe de type Cayley d'un système de réécriture décroissant simple.

Graphes de transductions contextuelles incrémentales. Nous montrons enfin qu'il est possible de donner une définition des graphes linéairement bornées comme les graphes de calcul d'une certaine famille de LBM, ou en d'autres termes comme les graphes définis par une certaine famille de transductions contextuelles (cf § 2.1.1.1 pour la définition des graphes de calcul).

Une LBM M réalise une relation binaire R si le langage de M est de la forme $\{u\#v \mid (u,v) \in R\}$ où $\#$ est un nouveau symbole. Cependant, ce type de transductions engendre plus que les graphes linéairement bornés. Même si l'on se restreint à des relations linéaires, (c'est à dire des relations R telles qu'il existe

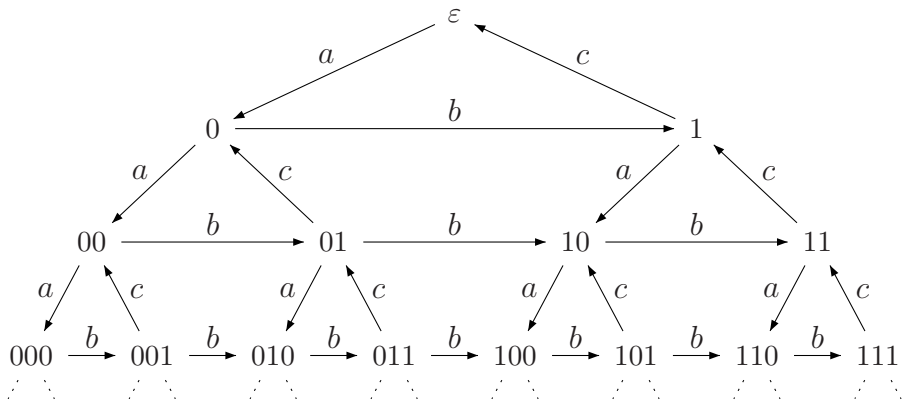


FIG. 4.13 – *Graphe de type Cayley du système décroissant* $R = \{a \to 0, b \to b, 0b \to 1, 1b \to b0, c \to c, 1c \to \varepsilon\}$.

---

c et $k \in \mathbb{N}$ tels que $(u,v) \in R$ implique $|v| \leq c \cdot |u| + k$), on obtient des graphes acceptant les langages reconnaissables en espace exponentiel (NSPACE[2^n]), qui incluent strictement les langages contextuels (cf. théorème 1.12). Il nous faut considérer des relations pour lesquelles la différence de longueur entre un mot et son image est bornée par une certaine constante. De telles relations peuvent être associées à des LBM de la façon suivante.

Définition 4.48. Une transduction contextuelle *k-incrémentale* T sur Γ est définie par une LBM acceptant le langage $L \subseteq \{u\#v \mid u,v \in \Gamma^* \text{ et } |v| \leq |u| + k\}$ où $\#$ n'appartient pas à Γ. La relation T est définie comme $\{(u,v) \mid u\#v \in L\}$.

Les relations rationnelles synchronisées d'image finie (c'est à dire telles que pour tout v il existe un nombre fini de u tels que $(u,v) \in R$) fournissent un premier exemple de transduction contextuelle k-incrémentale.

Proposition 4.49. *Pour toute relation rationnelle synchronisée R d'image finie, il existe une constance $k \in \mathbb{N}$ telle que R est une transduction contextuelle k-incrémentale.*

Démonstration. Soit $R \subseteq \Gamma^* \times \Gamma^*$ une relation rationnelle synchronisée d'image finie. Il découle de la définition de telles relations que R est égal à une union finie de la forme

$$\left(\bigcup_{i \in I} S_i \cdot (A_i \times \varepsilon)\right) \cup \left(\bigcup_{j \in J} S_j \cdot (\varepsilon \times B_j)\right)$$

où pour tout $k \in I \cup J$, S_k est une relation rationnelle synchrone, et pour tout $i \in I$ et $j \in J$, A_i et B_j sont des parties rationnelles de Γ^*.

Comme R est d'image finie, pour tout $j \in J$ l'ensemble B_j doit être fini. Soit k la longueur maximale des mots de $\bigcup_{j \in J} B_j$, il est très facile de vérifier que pour tout

couple de mots $(u,v) \in R$, $|v| \leq |u| + k$. De plus le langage $\{u\#v \mid (u,v) \in T\}$ est contextuel. Donc T est une transduction contextuelle k-incrémentale. □

La proposition suivante exprime le fait que les transductions contextuelles incrémentales d'un niveau donné forment une algèbre de Boole.

Proposition 4.50. *Pour toutes transductions contextuelles k-incrémentales T et T' sur Γ^*, $T \cup T'$, $T \cap T'$ et $\overline{T} = E_k - T$ (où E_k est $\{(u,v) \mid 1 \leq |v| \leq |u| + k\}$) sont des transductions contextuelles k-incrémentales.*

Démonstration. La clôture par union découle de celle des langages contextuels. Pour ce qui est du complémentaire, prenons $T \subset \Gamma^* \times \Gamma^*$ une transduction contextuelle k-incrémentale. Par définition, l'ensemble $L = \{u\#v \mid (u,v) \in T\}$ est contextuel. On peut facilement vérifier que l'ensemble $L' = \{u\#v \mid (u,v) \in E_k - T\}$ est égal à

$$\overline{L} \cap \{u\#v \mid |v| \leq |u| + k\}.$$

Comme les langages contextuels sont fermés à la fois par complémentaire (cf. théorème 1.11) et par intersection, L' est un langage contextuel. Par conséquent $\overline{T}$ est une transduction contextuelle k-incrémentale. □

On considère comme de coutume la fermeture par isomorphisme de la famille des graphes de transductions contextuelles incrémentales.

Exemple 4.51. Le graphe linéairement borné de la figure 4.12 peut être vu comme le graphe de transduction de l'ensemble de transductions contextuelles $\{T_a,T_b\}$ avec

$$T_a = \{(\#a^n,\#a^{n+1}) \mid n \geq 0\} \cup \{(b^m a^n,b^{m-1}a^{n+1}) \mid m \geq 1,n \geq 0\},$$
$$T_b = \{(\#a^n,a^{n-1}b) \mid n \geq 1\} \cup \{(a^m b^n,a^{m-1}b^{n+1}) \mid m \geq 1,n \geq 0\}.$$

En ce qui concerne le graphe de type Cayley de l'exemple 4.47, il peut être vu comme le graphe de transduction de l'ensemble de relations contextuelles incrémentales $\{T_a,T_b,T_c\}$, où T_a ajoute un 0 et T_c enlève un 1 à la droite d'un nombre écrit en binaire, et T_b implémente l'incrément en binaire.

Les transductions contextuelles préservant la longueur ont déjà été étudiées en détail dans [LST98]. Dans le reste de cette présentation, sauf mention expresse du contraire, nous nous restreindrons à des transductions contextuelles 1-incrémentales sans que cela ne restreigne la famille de graphes considérée.

Équivalence des trois définitions Nous démontrons maintenant que les familles des graphes de type Cayley de systèmes décroissants et des graphes de transductions contextuelles incrémentales coïncident toutes deux avec la famille des graphes linéairement bornés, à isomorphisme près.

Théoreme 4.52. *Pour tout graphe G, les énoncés suivants sont équivalents :*

1. *G est isomorphe au graphe de transition d'une LBM,*

2. *G est isomorphe au graphe de type Cayley d'un système de réécriture décroissant fini,*
3. *G est isomorphe à un graphe de transduction contextuel incrémental.*

Démonstration. $1 \implies 2$: Soit $M = (Q,\Sigma,\Gamma,\delta,q_0,F,[,])$ une machine linéairement bornée normalisée, avec $\Sigma \cap \Gamma = \emptyset$. Comme M est normalisée ses états de contrôle peuvent être partitionnés en deux ensembles Q_Σ et Q_ε (cf. § 2.1.1.1). Soit $\Gamma' = \Gamma \cup \{[,]\}$. Nous construisons un système de réécriture fini décroissant R dont le graphe de type Cayley est le graphe de transition de M. Soit $\Delta = \Sigma \cup \Gamma' \cup (\Gamma' \times Q) \cup S$ l'alphabet de R, avec $S = \{v_a,v'_a,s_a \mid a \in \Sigma\}$ un nouvel ensemble de symboles disjoint de Γ et Q. Les éléments (x,q) de $\Gamma' \times Q$ seront notés x_q. Par commodité, pour tout ensemble X, on notera $X_\bullet$ l'ensemble $X \cup (X \times Q)$, et $x_\bullet$ un symbole quelconque dans $\{x\}_\bullet$.

Les règles de R doivent permettre d'assurer plusieurs points importants :

1. Seuls des mots de la forme $(\varepsilon \cup [_\bullet)\Gamma^*_\bullet(\varepsilon \cup]_\bullet)$ doivent être des configurations stables (formes normales) :

$$x[_\bullet \to [_\bullet, \qquad]_\bullet y \to]_\bullet, \qquad s \to s$$

pour tous $x \in \Delta$, $y \in \Delta \setminus \Sigma$, $s \in \Sigma \cup S \cup (\Gamma' \times Q_\varepsilon)$.

2. Quand une lettre de Σ à la droite d'un mot irréductible u, il faut s'assurer que u représente effectivement une configuration légale, c'est à dire que u est d'une des formes $[_q\Gamma^*]$, $[\Gamma^* A_q \Gamma^*]$ ou $[\Gamma^*]_q$ pour $A \in \Gamma$, $q \in Q$:

$$]a \to v_a], \qquad]_q a \to {v_a}']_q, \qquad A_q v_a \to {v_a}' A_q,$$
$$A{v_a}' \to {v_a}' A, \qquad [{v_a}' \to [s_a$$

pour tous $a \in \Sigma$,$q \in Q$,$A \in \Gamma$.

3. Enfin, une fois vérifié que le mot représente bien une configuration de LBM, il faut simuler une opération d'insertion de cellule suivie d'un nombre quelconque d'ε-transitions de la machine :

$$s_a A \to A s_a, \qquad s_a B_p \to C_q B$$

pour tous $a \in \Sigma$, A,B,$C \in \Gamma$ et $pB \xrightarrow{a} qCB \in \delta$, et

$$A_p C \to B C_q, \qquad C A_p \to C_q B \qquad A_p C \to C_q$$

pour tous $pA \xrightarrow{\varepsilon} qB+$, $pA \xrightarrow{\varepsilon} qB-$, $pA \xrightarrow{\varepsilon} q \in \delta$ (respectivement).

Il existe un arc $u \xrightarrow{a} v$ dans le graphe de type Cayley G_R de R si et seulement si u et v sont des mots représentant des configurations valides de M depuis lesquelles aucune ε-transition ne peut être effectuée, c'est à dire des configurations externes,

et s'il existe une séquence de transitions étiquetée par $a\varepsilon^*$ dans M par laquelle u peut atteindre v. Il y a une bijection entre les arcs et les sommets de G_R et ceux du graphe de transition de M, donc ces deux graphes sont isomorphes.

2 $\Longrightarrow$ 3: Soit R un système de réécriture décroissant fini, G_R son graphe de type Cayley. Pour toute lettre a, nous allons montrer que la relation

$$T_a \;=\; \{(ua,v) \mid u \xrightarrow[G_R]{a} v\} = \{(ua,v) \mid u,v \in \mathrm{NF}(R) \wedge ua \xrightarrow[R]{*} v\}.$$

est une transduction contextuelle incrémentale en construisant la LBM $M_a = (Q,\Sigma,\Gamma,\delta,q_0,F,[,])$ acceptant T_a.

Pour tout couple (ua,v), M_a démarre dans la configuration $ua\#v$, et commence par vérifier que u est une forme normale pour R en s'assurant qu'il ne contient aucune partie gauche de règle de R. Deuxièmement, M_a simule la dérivation de R sur u, appliquant les règles de réécriture une par une jusqu'à ce qu'une forme normale soit atteinte. En raison du non-déterminisme, il peut exister des calculs non réussis, mais l'important est que la paire soit acceptée si et seulement si un calcul au moins atteint la configuration $v\#v$, signifiant que R peut normaliser ua en v. Donc un couple (ua,v) appartient à T_a si et seulement si $(u,a,v) \in G_R$, ce qui signifie que le graphe de transduction de $(T_a)_{a\in\Sigma}$ est isomorphe à G_R.

3 $\Longrightarrow$ 1: Soit $T = (T_a)_{a\in\Sigma}$ un ensemble fini de transductions contextuelles incrémentales définissant un graphe G_T, chaque T_a étant reconnu par une LBM M_a. Nous décrivons de manière informelle une LBM normalisée M dont le graphe de transition est isomorphe à G_T.

Soit q l'unique état de contrôle externe de M, M doit avoir un calcul étiqueté par $a\varepsilon^*$ entre les configurations qu et qv à chaque fois que $(u,v) \in T_a$, ou de façon équivalente si le mot $u\#v$ est accepté par M_a. Nous procédons de la manière suivante. Premièrement, partant de la configuration qu, M doit effectuer une transition étiquetée par a, augmentant son espace de bande disponible d'une case, et passer dans un état de contrôle interne. Elle doit ensuite deviner un mot v et écrire $u\#v$ sur sa bande. Comme T_a est incrémentale, ceci peut être fait en utilisant au plus $|u|+1$ cellules en écrivant deux symboles par cellule. Ensuite, M simule la machine M_a sur le mot d'entrée $u\#v$, tout en conservant une copie intacte de v sur la bande (une fois encore, ceci peut être réalisé par un simple codage sur l'alphabet). Si le calcul simulé de M_a réussit, M repasse dans la configuration externe qv en restaurant sur la bande sa copie sauvegardée de v, et dans le cas contraire elle boucle sur un état interne non-acceptant. Par cette construction, il existe un arc (qu,a,qv) dans G_M si et seulement si (u,a,v) appartient à G_T, et donc les deux graphes sont isomorphes. □

Ceci montre que les trois familles de graphes présentées dans ce paragraphe définissent en fait le même ensemble de graphes à isomorphisme près, à savoir les

graphes linéairement bornés. Cette diversité de définitions nous permettra dans la suite de démontrer de façon plus simple certaines propriétés de ces graphes.

4.3.2 Propriétés structurelles

À présent que les graphes linéairement bornés ont été définis à l'aide de trois formalismes différents, nous pouvons aisément en déduire certaines de leurs propriétés structurelles. En particulier, Nous nous intéressons aux langages acceptés par les graphes linéairement bornés ainsi qu'à certaines de leurs propriétés de clôture. Nous donnons aussi quelques éléments de réflexion sur le rapport entre les graphes linéairement bornés et les langages contextuels déterministes, et concluons par quelques propriétés logiques. Mais tout d'abord, nous comparons notre notion avec les travaux liés.

4.3.3 Comparaison aux travaux existants

Nous donnons ici une comparaison précise des graphes linéairement bornés avec les restrictions des graphes de Turing ([Cau03]) au cas linéairement borné, ainsi qu'aux graphes des configurations considérés dans [KP99].

Graphes des configurations. Dans [KP99], les graphes des configurations d'une famille de machines linéairement bornées très semblable à nos LBM étiquetées sont considérées. Cependant, leur définition inclut une restriction à l'ensemble des configurations accessibles depuis la configuration initiale. Par conséquent, cette notion du graphe des configurations d'une machine est incomparable à la nôtre. Comme nous le verrons à la proposition 4.56, les graphes linéairement bornés sont clos par restriction à l'ensemble des sommets accessibles depuis un sommet donné. Ainsi il n'est pas nécessaire d'imposer de restriction par accessibilité directement dans la définition des graphes des configurations.

Mise à part cette restriction, notre famille de graphes de configurations, coïncide avec les graphes de configurations de [KP99] ainsi qu'aux graphes de configurations de [Cau03] dans le cas linéairement borné.

Graphes de transition. Knapik et Payet ne considèrent pas dans leur travail de réelle notion de graphes de transitions. Au lieu de définir l'ε-clôture du graphe des configurations, ils démontrent une propriété de clôture de cette famille *à bisimulation faible près* [Mil89], qui n'est pas caractérisée de façon structurelle. Néanmoins, il est peu difficile de montrer que leurs résultats peuvent être étendus aux graphes de transition pris à isomorphisme près, comme il en est traité au paragraphe 4.3.2.

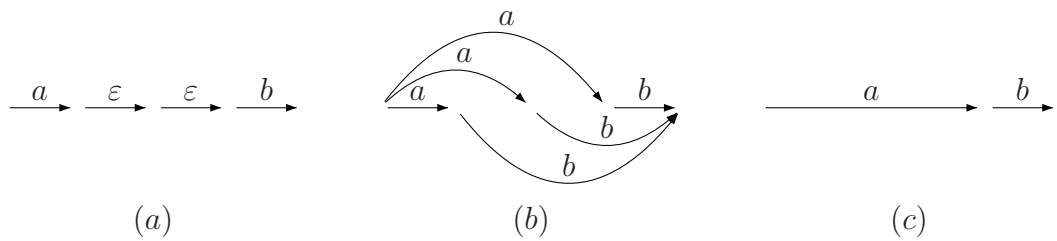


FIG. 4.14 – *(a) Graphe des configurations d'une LBM normalisée. (b) Graphe de transition de (a) tel que défini dans [Cau03] (sans restriction sur les sommets). (c) Graphe linéairement borné associé à (a).*

Dans [Cau03], Caucal définit les graphes de transition des machines de Turing à partir de leurs graphes de configurations grâce à la notion d'ε-clôture : partout où il existe un chemin étiqueté par $\varepsilon^* a \varepsilon^*$ dans un graphe de configurations, il y a un arc étiqueté par a dans son ε-clôture, pour toute lettre a. De plus, la définition prévoit une restriction à un ensemble rationnel quelconque de sommets (potentiellement l'ensemble de tous les sommets). La figure 4.14 illustre sur un petit graphe de configurations la différence entre cette approche et la nôtre.

Notre approche a deux avantages principaux. Tout d'abord, l'opération d'ε-clôture telle qu'elle est définie par Caucal peut faire apparaître du non-déterminisme artificiel dans le graphe obtenu. Cette complexité structurelle supplémentaire n'a pas lieu d'être et est éliminée dans notre définition. De plus, notre notion est purement structurelle, et ne s'appuie pas sur le nommage des sommets. Mais il est intéressant de remarquer que les deux familles de graphes coïncident malgré tout à isomorphisme près : pour toute machine linéairement bornée M et tout ensemble rationnel R, on peut définir une machine M' telle que le graphe de transition de M suivant la notion de Caucal est isomorphe au graphe linéairement borné défini par M' dans notre cadre.

Proposition 4.53. *La famille des graphes linéairement bornés coïncide, à isomorphisme près, avec la restriction au cas linéairement borné des graphes de Turing.*

Démonstration. Soit M une LBM normalisée, C son graphe des configurations, R son ensemble (rationnel) de configurations externes et G son graphe de transition défini comme

$$G = \{c \xrightarrow{a} c' \mid c,c' \in R \wedge c \xrightarrow{a\varepsilon^*} c' \in C\}.$$

Considérons le graphe G' défini comme la restriction à R de l'ε-clôture de C.

$$G' = \{c \xrightarrow{a} c' \mid c,c' \in R \wedge c \xrightarrow{\varepsilon^* a \varepsilon^*} c' \in C\}.$$

Les graphes G et G' sont égaux, puisque les configurations externes n'ont par définition aucune ε-transition sortante.

Réciproquement, soit M une LBM (pas forcément normalisée), C son graphe de configurations, et R un ensemble rationnel de configurations de M. La relation binaire $\xrightarrow{\varepsilon^* a \varepsilon^*}$ incluse dans $C \times C$ est une transduction contextuelle 1-incrémentale. Par conséquent le graphe G défini comme la restriction à R (cf. proposition 4.56) de l'ε-clôture de C est un graphe linéairement borné. □

4.3.3.1 Langages

Il est relativement évident que le langage du graphe de transition d'une LBM M entre le sommet représentant se configuration initiale et l'ensemble des sommets représentant ses configurations finales est le langage de M lui-même. En fait, on peut montrer que le libre choix du sommet initial et des sommets finaux n'a pas d'influence sur la famille de langages obtenue.

Proposition 4.54. *Les langages des graphes linéairement bornés entre un sommet initial i et un ensemble fini F de sommets finaux sont les langages contextuels.*

Démonstration. Soit G un graphe linéairement borné étiqueté sur Σ défini par une famille $(T_a)_{a\in\Sigma}$ de transductions contextuelles 1-incrémentales. Nous allons démontrer que $L(G,i,F)$ est contextuel même si F est un ensemble contextuel.

Soit $\# \notin \Sigma$ un nouveau symbole, nous considérons le graphe $\overline{G}$ obtenu depuis G en ajoutant une boucle étiquetée par $\#$ à chaque sommet de l'ensemble F. Il est évident que, $\overline{G}$ est linéairement borné puisque $\xrightarrow{\#} = \{(f,f) \mid f \in F\}$ est une transduction contextuelle 0-incrémentale. D'après le théorème 4.52, G est le graphe de transition d'une LBM M. Soit c_0 la configuration de M correspondant au sommet i dans G. Considérons la LBM M' dont la configuration initiale est i (voir la remarque 1.7) et dont les états finaux sont les états apparaissant dans la partie gauche d'une règle étiquetée par $\#$ (on peut supposer que l'existence d'une règle étiquetée par $\#$ ne dépend pas du contenu actuel de la cellule). On peut facilement vérifier que M' accepte $L(G,i,F)$, qui est donc contextuel.

Pour la réciproque, soit L un langage contextuel et M une LBM normalisée acceptant L. Le graphe de transition de M accepte L depuis sa configuration initiale jusqu'à l'ensemble (infini) de ses configurations acceptantes. Pour ramener cet ensemble de configurations à un ensemble fini, il suffit d'ajouter un nouvel état de contrôle externe q_f à la machine M, et à chaque fois que M effectue une ϵ-transition vers un état final, on ajoute la possibilité d'atteindre la configuration $[q_f]$ par une séquence d'ϵ-transitions. Comme $[q_f]$ n'a pas de transitions sortantes, c'est une configuration externe et il est facile de voir que $L = L(G,c_0,[q_f])$. □

Remarque 4.55. Quand un graphe linéairement borné est donné explicitement comme le graphe de transition d'une LBM, un graphe de type Cayley ou un

graphe de transduction, c'est à dire quand le nommage de ses sommets est fixé, considérer un ensemble contextuel de sommets finaux n'ajoute pas à la famille de langages obtenue.

4.3.3.2 Propriétés de clôture

Les graphes linéairement bornés jouissent de plusieurs propriétés intéressantes, qui seront utile lors de la comparaison de cette famille avec d'autres familles de graphes liées aux langages contextuels (cf. § 4.4).

Proposition 4.56. *La famille des graphes linéairement bornés est close par restriction à l'ensemble des sommets accessibles depuis un sommet quelconque et par restriction à un ensemble contextuel de sommets.*

Démonstration. Nous commençons par démontrer la fermeture par restriction aux sommets accessibles. Soit G un graphe linéairement borné défini par une famille $(T_a)_{a\in\Sigma}$ de transductions contextuelles 1-incrémentales et soit u_0 un sommet de V_G. Chaque transduction T_a pour $a \in \Sigma$ est acceptée par une LBM M_a. Considérons le graphe G' obtenu en restreignant le graphe G à l'ensemble de ses sommets accessibles depuis u_0. Notons que cet ensemble est en général non contextuel, comme le montre l'exemple suivant.

Exemple 4.57. Soit un langage $L \subset \{a,b\}^*$ reconnaissable en temps exponentiel mais pas linéaire (c'est à dire $L \in \text{NSPACE}[2^n] \setminus \text{NSPACE}[n]$). L'existence d'un tel langage est garantie par le théorème 1.12. Considérons le graphe linéairement borné G étiqueté par $\{a,b,\sharp\}$ et défini par :

- $\sharp u \xrightarrow[G]{x} \sharp ux$ pour tous $u \in \{a,b\}^*$ et $x \in \{a,b\}$,
- $\sharp u\sharp^n \xrightarrow[G]{\sharp} \sharp u\sharp^{n+1}$ pour tous $u \in \{a,b\}^*$ et $n+1 \leq 2^{|u|}$,
- et $\sharp u\sharp^{2^{|u|}} \xrightarrow[G]{\sharp} u$ pour tout $u \in L$.

La figure 4.15 illustre la construction de ce graphe. L'ensemble des sommets accessibles depuis $\sharp$ restreint à $\{a,b\}^*$ est exactement le langage L, qui par définition n'est pas contextuel.

Par conséquent, nous devons adopter un nommage différent des sommets de G'. Nous construisons un graphe H isomorphe à G' dans le but de prouver que H, et donc aussi G', sont linéairement bornés. Pour tout couple de sommets u et $v \in V_G$, on note $u \Rightarrow_n v$ s'il existe un chemin de u à v dans G n'utilisant que des sommets de taille inférieure ou égale à n. L'ensemble des sommets de H est $V_H = \{u\square^n \mid u \in \Gamma^*, u_0 \Rightarrow_{|u|+n} u \text{ and } u_0 \not\Rightarrow_{|u|+n-1} u\}$. Remarquons que pour tout $u \in V_G$, il existe un unique n tel que $u\square^n \in V_H$. Pour tout $a \in \Sigma$, on prend $u\square^n \xrightarrow[H]{a} v\square^m$ si et seulement si $u\square^n \in V_H$, $v\square^m \in V_H$ et $u \xrightarrow[G]{a} v$.

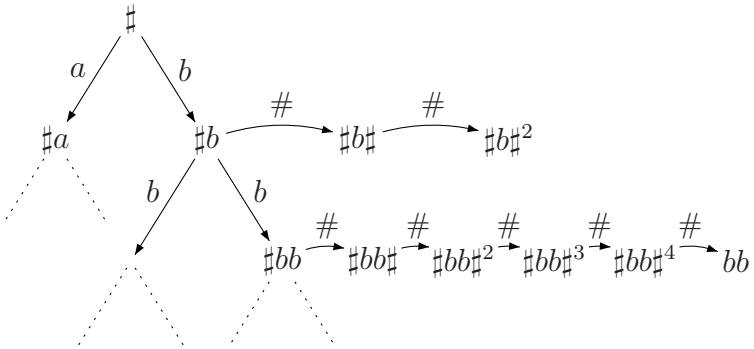


FIG. 4.15 – *Le graphe G pour un certain L contenant bb mais pas b.*

Le graphe H est isomorphe à G'. On peut vérifier que l'application ρ de $V_{G'}$ dans V_H associant à tout $u \in V_{G'}$ l'unique $u\square^n \in V_H$ est une bijection et un morphisme de graphes de G' dans H. Nous démontrons d'abord que V_H est contextuel et que pour tout $a \in \Sigma$, $\xrightarrow[H]{a}$ est une transduction contextuelle 1-incrémentale. Il s'ensuivra que H et G' sont des graphes linéairement bornés.

1. Considérons le langage L^+ égal à $\{u\square^n \mid u_0 \Rightarrow_{|u|+n} u\}$ et le langage L^- égal à $\{u\square^n \mid u_0 \Rightarrow_{|u|+n-1} u\}$, on peut vérifier que $V_H = L^+ \cap \overline{L^-}$. Si l'on démontre que L^+ et L^- sont contextuels, il en découlera d'après le théorème 1.11 que V_H est un ensemble contextuel.
   Nous construisons une LBM M acceptant L^+. Partant du mot $u\square^n$, M devine un chemin de u_0 à u dont les sommets sont de taille au plus $|u|+n$. Le calcul commence par u_0, devine un mot u_1 de longueur au plus $|u|+n$ puis simule l'une des machines M_a pour vérifier que $u_0 \xrightarrow[G]{a} u_1$. Enfin, M remplace u_0 par u_1 et itère le processus jusqu'à ce que u_i soit égal à u. Ceci utilise au plus $3(|u|+n)$ cellules. Une construction semblable permet de reconnaître L^-.
2. Pour tout $a \in \Sigma$, $u\square^n \xrightarrow[H]{a} v\square^m$ implique que $|v| \leq |u|+1$ comme $u \xrightarrow[G]{a} v$ et $\xrightarrow[G]{a}$ est 1-incrémentale, et $m \leq n$ comme $u_0 \Rightarrow_n u$ et $u \xrightarrow[G]{a} v$. Par conséquent, $|v|+m \leq |u|+n+1$ et donc $\xrightarrow[H]{a}$ est 1-incrémentale.
   Le langage $L_a = \{u\square^n\sharp v\square^m \mid u\square^n \xrightarrow[H]{a} v\square^m\}$ est accepté par une LBM qui vérifie que $u\square^n$ et $v\square^m$ appartiennent à V_H (en simulant une machine acceptant V_H) et que $u \xrightarrow[G]{a} v$. Donc $\xrightarrow[H]{a}$ est une transduction contextuelle 1-incrémentale.

Exemple 4.58. Cette construction appliquée au graphe de l'exemple 4.57 donne

le graphe H défini par

- $\sharp u \xrightarrow[G]{x} \sharp ux$ pour tous $u \in \{a,b\}^*$ et $x \in \{a,b\}$,
- $\sharp u\sharp^n \xrightarrow[G]{\sharp} \sharp u\sharp^{n+1}$ pour tous $u \in \{a,b\}^*$ et $n+1 \leq 2^{|u|}$,
- et $\sharp u\sharp^{2^{|u|}} \xrightarrow[G]{\sharp} u\Box^{2^{|u|}}$ pour tout $u \in L$.

Démontrons maintenant la clôture des graphes linéairement bornés par restriction à un ensemble contextuel de sommets. Soit G un graphe linéairement borné défini par une famille $(T_a)_{a\in\Sigma}$ de transductions contextuelles 1-incrémentales. D'après la proposition 4.50, la transduction $T_a \cap \{(u,v) \mid u \in F, v \in F \text{ and} |v| \leq |u|+1\}$ est également contextuelle et 1-incrémentale. Par conséquent $G|_F$ est un graphe linéairement borné. □

Puisque tous les langages rationnels sont contextuels, les graphes linéairement bornés sont également clos par restriction rationnelle. Ceci montre qu'il n'est pas nécessaire de prendre un compte des restrictions rationnelles arbitraires ou des restrictions par accessibilité dans la définition des graphes de transition de machines linéairement bornées, puisque de telles restrictions peuvent être codées directement dans la machine.

Pour finir ce paragraphe, nous étendons le résultat de [KP99] à la classe des graphes linéairement bornés, et montrons que cette classe est fermée par produit synchronisé [AN82].

Proposition 4.59. *La famille des graphes linéairement bornés est fermée par produit synchronisé.*

Démonstration. Le produit synchronisé de deux graphes G et G' à étiquettes dans Σ et Σ' par rapport à un ensemble de contraintes $C \subseteq \Sigma \times \Sigma'$ est le graphe

$$G \otimes G' = \{(u,v) \xrightarrow{(a,b)} (u',v') \mid u \xrightarrow[G]{a} u' \wedge v \xrightarrow[G]{b} v' \wedge (a,b) \in C\}.$$

On peut directement déduire de cette définition que si G et G' sont tous deux linéairement bornés, et définis par exemple comme les graphes de deux familles de transductions contextuelles incrémentales, les relations d'arcs de leur produit synchronisé sont elles aussi des transductions contextuelles incrémentales. □

4.3.3.3 Graphes linéairement bornés déterministes

Nous considérons à présent les relations entre le déterminisme des graphes linéairement bornés et celui des machines linéairement bornées les définissant. Comme on l'a remarqué au paragraphe 2.1.1.1, il existe des machines non-déterministes, et en particulier des LBM, dont le graphe de transition est déterministe.

Plus précisément, toute machine dans laquelle, d'une configuration externe donnée, au plus une configuration externe est accessible par un calcul partiel étiqueté par $a\varepsilon^*$, où a est une étiquette quelconque, possède un graphe de transition déterministe. En fait, on peut montrer que toute LBM peut être transformée en une LBM équivalente possédant cette propriété. Par conséquent, comme l'exprime la proposition suivante, tout langage contextuel est accepté par au moins un graphe linéairement borné déterministe.

Proposition 4.60. *Pour tout langage contextuel L, il existe un graphe linéairement borné* déterministe *G, un sommet i et un ensemble rationnel F de sommets de G tels que $L = L(G,\{i\},F)$. De plus, on peut toujours choisir pour G un arbre infini.*

Démonstration. Soit $L \subset \Sigma^*$ un langage contextuel. Nous construisons un graphe linéairement borné G étiqueté par Σ dont les sommets sont des mots de la forme ε, Aw ou Rw, avec $w \in \Sigma^*$ et $A,R \notin \Sigma$ (où A signifie « accepter » et R « refuser »), et dont les arcs sont définis par les transductions contextuelles 1-incrémentales suivantes :

$$\begin{aligned} &Au \xrightarrow{a} Aua, \quad Rv \xrightarrow{a} Ava && \text{pour tous } u \in L, v \notin L \text{ et } ua, va \in L, \\ &Au \xrightarrow{a} Rua, \quad Rv \xrightarrow{a} Rva && \text{pour tous } u \in L, v \notin L \text{ et } ua, va \notin L. \end{aligned}$$

Le langage de ce graphe déterministe entre le sommet A si $\varepsilon \in L$, ou R si $\varepsilon \notin L$, et l'ensemble rationnel $A\Gamma^*$ est précisément le langage L. De plus, G est un arbre isomorphe à l'arbre complet étiqueté sur Σ. □

Remarque 4.61. La démonstration précédente ne fonctionne pas pour un ensemble fini de sommets finaux même si l'on ne considère que des langages contextuels déterministes. Considérons par exemple le langage $L = \{(a^n b)^* | n \in \mathbb{N}\}$. Supposons qu'il existe un graphe déterministe G tel que $L = L(G,i,F)$ pour un certain ensemble fini F. Alors il devrait exister $m \neq n$ tel que $i \xrightarrow[G]{a^n b} f$ et $i \xrightarrow[G]{a^m b} f$. Comme $f \xrightarrow[G]{a^m b} f'$ pour un certain $f' \in F$, il s'ensuivrait forcément que $i \xrightarrow[G]{a^n b a^m b} f'$.

Bien sûr, on ne peut pas conclure de ceci que les langages des graphes linéairement bornés déterministes sont les langages contextuels déterministes, ce qui reviendrait à répondre à la question générale posée par [Kur64] de savoir si les langages contextuels déterministes et non-déterministes coïncident. Cependant, si l'on ne considère que des machines linéairement bornées qui terminent (qui n'ont aucun calcul infini sur aucun mot d'entrée), alors la famille de graphes de transition que nous obtenons illustre fidèlement le déterminisme des langages.

Remarque 4.62. Même pour une LBM terminante, le déterminisme du graphe de transition n'implique pas nécessairement celui de la machine. La figure 4.16 illustre ce fait. L'idée de la démonstration qui va suivre est de montrer que toute

FIG. 4.16 – *Graphe des configurations et graphe de transition déterministe d'une LBM terminante non-déterministe.*

LBM terminante dont le graphe de transition est déterministe peut être déterminisée sans que la structure de graphe ne change (et donc que son langage est contextuel déterministe).

Proposition 4.63. *Les langages des graphes de transition déterministes de machines linéairement bornées déterministes terminantes (entre un sommet initial et un ensemble rationnel de sommets finaux) sont les langages contextuels déterministes.*

Démonstration. Soit L un langage contextuel déterministe. il existe une LBM déterministe terminante M qui accepte L. Par définition, son graphe de transition G_M est déterministe, et il accepte L entre le sommet correspondant à la configuration initiale et les sommets correspondant aux configurations finales de M.

Réciproquement, soit G_M le graphe de transition déterministe d'une LBM terminante $M = (Q,\Sigma,\Gamma,\delta,q_0,F,[,])$. Nous construisons une LBM déterministe N dont le graphe de transition G_N est égal à G_M, et surtout dont le langage est le même que celui de M. La machine N s'obtient depuis M en conservant exactement une règle de δ pour chaque partie gauche et étiquette donnée. Elle est définie comme $N = (Q,\Sigma,\Gamma,\delta',q_0,F,[,])$ où δ' est un sous-ensemble de quelconque de δ tel que

- si $pA \xrightarrow{x} qU \in \delta$ alors il existe $pA \xrightarrow{x} q'U' \in \delta'$,
- si $pA \xrightarrow{x} qU \in \delta'$ alors pour tout $pA \xrightarrow{x} q'U' \in \delta'$, $q = q'$ et $U = U'$.

Par construction, N est déterministe. De plus, l'ensemble de configurations externes de N est égal à celui de M. Il reste à démontrer que pour tout couple de configurations externes c et c', $c \xrightarrow[G_M]{a} c'$ si et seulement si $c \xrightarrow[G_N]{a} c'$.

Si $c \xrightarrow[G_M]{a} c'$, il existe un chemin entre c et c' étiqueté par $a\varepsilon^*$ dans le graphe de configurations C_M de M. Comme G_M est déterministe et terminante, tout chemin maximal de C_M étiqueté par $a\varepsilon^*$ se termine en c'. Cette propriété reste vraie pour le graphe des configurations C_N. Supposons pas l'absurde qu'il existe un chemin dans C_N étiqueté par $a\varepsilon^*$ et d'origine c dont la destination serait un sommet

externe $c'' \neq c'$. Ceci impliquerait que soit c'' ait des arcs sortants étiquetés par ϵ dans C_M et pas dans C_N, soit que c'' soit une configuration externe de M différente, ce qui est impossible par construction de N.

Réciproquement si $c \xrightarrow[G_N]{a} c'$ alors, par construction de N, nous avons $c \xrightarrow[G_M]{a} c'$. Donc $G_M = G_N$. □

Remarque 4.64. D'autres définitions équivalentes des graphes de transition déterministes de LBM terminantes peuvent être données : ils coïncident avec les graphes de type Cayley des systèmes décroissants fortement normalisants, ainsi qu'avec les graphes des transductions incrémentales déterministes terminantes.

Remarque 4.65. Il est bien sûr impossible d'utiliser les constructions précédentes pour déterminiser une machine linéairement bornée quelconque. En effet, la construction de la proposition 4.60 produit des machines non-terminantes en général, et la construction de la proposition 1.10 ne préserve pas la structure du graphe de transition d'une machine.

4.3.3.4 Propriétés logiques

Pour terminer ce paragraphe sur les propriétés des graphes linéairement bornés, nous nous intéressons à la décidabilité de la théorie du premier ordre des graphes de configuration de LBM et des graphes linéairement bornés. En raison de la grande expressivité du modèle considéré, seules les propriétés locales exprimées en logique du premier ordre sont décidables sur un graphe de configurations de LBM.

Proposition 4.66. *Les graphes de configuration de machines de Turing linéairement bornées ont une théorie du premier ordre décidable.*

Démonstration. Il s'agit d'une conséquence directe du fait que les graphes de configuration des LBM sont des graphes rationnels synchronisés, dont la théorie du premier ordre est décidable [BG00]. □

Cependant, comme remarqué par [KP99], il n'existe pas l'algorithme qui, étant donné une LBM M et une formule close du premier ordre ϕ, décide si le graphe de transition de M satisfait ϕ. Ce constat peut être renforcé sous la forme de la proposition suivante.

Proposition 4.67. *Il existe un graphe linéairement borné de théorie du premier ordre indécidable.*

Démonstration. Considérons une énumération $(M_n)_{n\in\mathbb{N}}$ de toutes les machines de Turing (non étiquetées), et une machine de Turing étiquetée M dont le langage est l'ensemble de tous les mots de la forme $\#^n$ tels que la machine M_n termine sur le mots vide. À l'aide uniquement d'ε-transitions, M devine un nombre n et

écrit $\#^n$ sur sa bande, puis simule la machine numéro n sur l'entrée vide. Si la machine s'arrête, alors M lit le mot $\#^n$ et accepte. Tous les calculs acceptants de M sont donc étiquetés par $\varepsilon^*\#^n$ pour un certain n.

Si l'on remplace ε par un symbole observable τ, le graphe de configuration C de M est un graphe linéairement borné. Soit C' sa restriction par accessibilité depuis la configuration initiale de M. Le graphe C' est toujours linéairement borné d'après la proposition 4.56. Pour tout n, la formule ϕ_n exprimant l'existence d'un chemin étiqueté par $\tau\#^n$ et se terminant dans un sommet sans successeur est satisfaite dans C' si et seulement si la machine M_n s'arrête sur l'entrée vide ε. L'ensemble de telles formules satisfaisables n'est pas récursif, par conséquent la théorie du premier ordre de C' est indécidable (et pas même récursivement énumérable). □

4.4 Comparaison des deux familles

Nous donnons à présent quelques remarques sur la comparaison des graphes linéairement bornés et plusieurs sous-familles des graphes rationnels. Notons tout d'abord que comme les graphes linéairement bornés ont par définition un degré sortant fini, il n'est significatif de les comparer qu'aux graphes rationnels de degré sortant fini. Cependant, même sous cette contrainte structurelle on se rend compte que les graphes rationnels et linéairement bornés sont incomparables, à cause de différences dans la croissance du degré de leurs sommets.

Une première observation est que le degré sortant des sommets d'un graphe rationnel à la distance n d'un sommet quelconque peut être aussi grand que c^{c^n} pour une certaine constance c dépendant du graphe et du sommet choisi (cf. Proposition 2.9), tandis que dans un graphe linéairement borné il est au plus de c^n.

Lemme 4.68. *Pour tout graphe linéairement borné G et tout sommet x, il existe $c \in \mathbb{N}$ tel que le degré sortant des sommets de G à une distance $n > 0$ de x est d'au plus c^n.*

Démonstration. Soit $(T_a)_{a\in\Sigma}$ un ensemble de transductions contextuelles incrémentales décrivant G et soit $k_a \in \mathbb{N}$ l'entier tel que T_a est une transduction k_a-incrémentale. On pose k le maximum de $\{k_a \mid a \in \Sigma\} \cup \{|x|\}$. À la distance $n > 0$ de x, un sommet a une longueur d'au plus $k(n+1)$, et donc son degré sortant est borné par le nombre de sommets de longueur au plus $k(n+2)$, qui est inférieur à $|\Gamma|^{k(n+2)+1}$. Donc il existe $c \in \mathbb{N}$ tel que le degré de tout sommet à distance n de x est borné par c^n. □

La figure 4.17 montre un graphe rationnel dont les sommets à la distance n de la racine A ont un degré sortant de $2^{2^{n+1}}$. Ce graphe n'est donc pas linéairement borné.

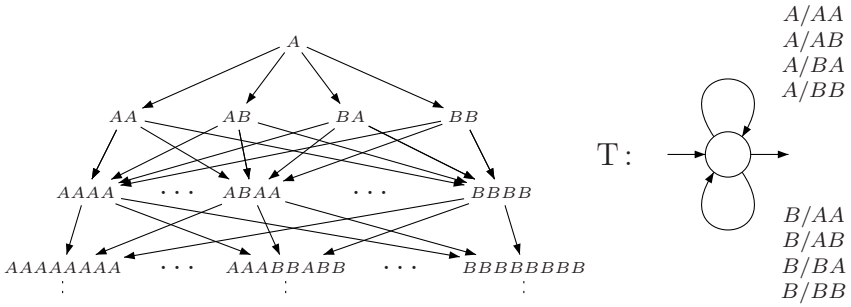


FIG. 4.17 – *Un graphe rationnel de degré fini (et les transducteurs le définissant) qui n'est isomorphe à aucun graphe linéairement borné.*

Réciproquement, dans un graphe rationnel de degré entrant fini le degré entrant de tout sommet à distance n d'un sommet désigné quelconque est au plus de c^{c^n} pour une certaine constante $c \in \mathbb{N}$. Dans un graphe linéairement borné, par contre, le degré entrant d'un tel sommet peut être aussi grand que $f(n)$ pour toute application f de $\mathbb{N}$ dans $\mathbb{N}$ reconnaissable en espace linéaire (c'est à dire telle que le langage $\{0^n 1^{f(n)} \mid n \in \mathbb{N}\}$ est contextuel).

Lemme 4.69. *Pour toute application $f : \mathbb{N} \longrightarrow \mathbb{N}$ reconnaissable en espace linéaire, il existe un graphe linéairement borné G et un sommet x de G tel que le degré entrant de tout sommet à distance $n > 0$ de x peut atteindre $f(n)$.*

Démonstration. Soit f une application de $\mathbb{N}$ dans $\mathbb{N}$ reconnaissable en espace linéaire, et soit G_f le graphe linéairement borné défini par les transductions contextuelles

$$\begin{aligned} T \;=\; & \{(u,u0) \mid u \in 0^*\} \;\cup\; \{(u,u1) \mid u \in 0^n 1^m,\, m < f(n)\} \\ & \cup \{(uv,u) \mid u \in 0^*, v \in 1^* \text{ and } |v| \leq f(|u|)\}. \end{aligned}$$

La figure 4.18 illustre la construction de G_f. Le degré entrant du sommet 0^n à la distance n de la racine ε est bien égal à $f(n)$. □

Il est aisé de construire une LBM reconnaissant le langage $\{0^n 1^m \mid m \leq f(n)\}$ pour $f(n) = 2^{2^{2^n}}$ par exemple, ce qui est incomparable avec les degrés entrants atteints dans un graphe rationnel de degré entrant fini. Nous avons établi qu'il existait un graphe linéairement borné qui n'est isomorphe à aucun graphe rationnel (qu'il soit de degré fini ou infini). On peut donc tirer de ces observations la conclusion suivante :

Proposition 4.70. *Les familles des graphes rationnels de degré sortant fini et des graphes linéairement bornées sont incomparables.*

En conséquence de ce résultat, nous nous intéressons à des sous-familles plus restreintes des graphes rationnels. En ce qui concerne les graphes synchronisés de

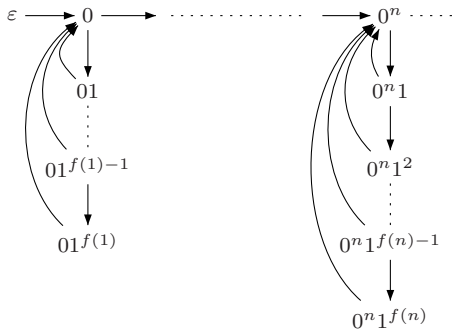


FIG. 4.18 – *Un graphe linéairement borné isomorphe à aucun graphe rationnel.*

degré fini, nous obtenons le résultat suivant :

Proposition 4.71. *Les graphes synchronisés de degré sortant fini sont une sous-famille stricte des graphes linéairement bornés.*

Démonstration. D'après la proposition 4.49, toute relation synchronisée d'image finie est contextuelle incrémentale. Donc, tout graphe synchronisé de degré sortant fini est un graphe linéairement borné. L'inégalité de ces familles peut se déduire (entre autres) du fait que les graphes synchronisés ne sont pas clos par restriction par accessibilité (cf. proposition 4.56). □

Pour la famille encore plus restreinte des graphes rationnels de degré borné, nous obtenons l'inclusion suivante.

Théoreme 4.72. *Les graphes rationnels de degré borné sont une sous-famille stricte des graphes linéairement bornés de degré borné (à isomorphisme près).*

Démonstration. Le résultat d'inclusion s'appuie sur le résultat d'uniformisation de Weber déjà utilisé précédemment (cf. proposition 4.35). Soit $G = (T_a)_{a\in\Sigma}$ un graphe rationnel sur Γ dont le degré sortant est borné par k. Chaque relation T_a est par conséquent k-valuée, et il existe donc k fonctions rationnelles $F_{a_1}, \dots F_{a_k}$ telles que $T_a = \bigcup_{i\in[1,k]} F_{a_i}$. On note $X = \{a_i \mid a \in \Sigma \text{ and } i \in [1,k]\}$.

Nous allons représenter chaque sommet x de G par un couple $(w,t) \in V_G \times X^*$ tel que $x = F_{t_{|t|}}(\dots(F_{t_1}(w)))$. Cependant, comme il peut exister plusieurs couples possibles pour un sommet donné, nous définissons un ordre total $<$ sur $V_G \times X^*$ et associons à chaque sommet x le plus petit couple possible selon cet ordre. Tout d'abord, soient $<_\Gamma$ et $<_X$ deux ordres totaux arbitraires sur Γ et X respectivement et soient $\prec_\Gamma$ et $\prec_X$ les ordres lexicographiques engendrés par $<_\Gamma$ et $<_X$. On pose

$$\begin{array}{lll} (m,t) < (n,r) & \text{ssi} & |m| + |t| < |n| + |r| \\ & \text{ou} & |m| + |t| = |n| + |r| \text{ et } m \prec_\Gamma n \\ & \text{ou} & |m| + |t| = |n| + |r|, m = n \text{ et } t \prec_X r \end{array}$$

On démontre facilement que $<$ est bien un ordre total sur $V_G \times X^*$. Nous allons maintenant démontrer que la fonction N qui associe à chaque couple (w,t) le plus petit couple (m,r) tel que $F_t(w)=F_r(m)$ est une transduction contextuelle 0-incrémentale.

Il y a deux points importants dans cette démonstration. Le premier est d'être capable de tester en espace linéaire pour deux couples (m,r) et (w,t) donnés si $F_r(m) = F_t(w)$. En d'autres termes, nous voulons démontrer que le langage $L = \{m\sharp r\$t\sharp w \mid F_r(m) = F_t(w)\}$ est contextuel.
Soit $\bar{X}$ un alphabet fini disjoint de X mais en bijection avec lui (pour tout $x \in X$, on note $\bar{x}$ le symbole correspondant de $\bar{X}$). Nous considérons le graphe rationnel $\bar{G}$ défini par la famille de transducteurs $(F_a)_{a\in X} \cup (\bar{F}_{\bar{a}})_{a\in X}$ où pour tout $\bar{a} \in \bar{X}$, $\bar{F}_{\bar{a}} = F_a^{-1}$. Il est aisé de vérifier que pour tous $m,w \in \Gamma^*$ et $r,t \in X^*$, il existe un chemin $m \xrightarrow[\bar{G}]{r\bar{t}} w$ si et seulement si $F_r(m) = F_t(w)$, avec $\bar{t} = \bar{t}_{|t|} \dots \bar{t}_1$. D'après le corollaire 4.19, le langage $\{i\sharp w\sharp f \mid w \in L(\bar{G},i,f)\} \cap \Gamma^*\sharp X^*\bar{X}^*\sharp\Gamma^*$ est contextuel. Il s'ensuit que L l'est aussi.

Le second point est d'effectuer ce test pour tous mes couples $(m,r) < (w,t)$. Même si le nombre de tels couples est fini, il se pose un problème si le test échoue. En effet, cela pourrait signifier soit que $m\sharp r\$t\sharp w \notin L$, soit que le chemin est effectivement correct et qu'il existe un autre calcul acceptant permettant de le vérifier. Ainsi, sur un test raté on ne peut pas directement passer au candidat (m,r) suivant.

Pour régler ce problème, une idée est de réaliser un test d'appartenance qu langage complémentaire $\overline{L}$ de L, qui est lui aussi contextuel. Dans le cas où le test sur le complémentaire échoue lui aussi, ce calcul pour $N(w,t)$ peut échouer sans danger, puisqu'on est assuré qu'au moins un autre calcul réussira soit pour le test d'appartenance à L soit pour son complémentaire. Dans le cas où le test sur $\overline{L}$ réussit, cela signifie que le couple (m,r) actuel n'est pas correct et que nous pouvons passer au couple suivant.

Le calcul de $N(w,t)$ réussit quand un couple (m,r) convenable est trouvé, et échoue quand tous les couples $(m,r) < (w,t)$ ont été testés et rejetés. N est 0-incrémentale puisque par définition de l'ordre total $<$, $(m,r) < (w,t) \implies |m| + |r| \leq |w| + |t|$, et que toutes les étapes de la construction sont réalisables en espace linéaire.

Nous pouvons à présent définir le graphe linéairement borné H à sommets dans $\Gamma^*\#X^*$ défini par l'ensemble de transductions $(T'_a)_{a\in\Sigma}$ défini comme suit :

$$T'_a = \bigcup_{i\in[1,k]} \{(w\#x,w'\#y) \mid N((w,x)) = (w,x) \text{ and } N(w,xa_i) = (w',y)\}.$$

Comme N est 0-incrémentale, T'_a est 1-incrémentale. L'application ρ de V_G dans $V_G \times X^*$ qui associe à tout sommet $x \in V_G$ le plus petit (w,t) tel que $x = F_t(w)$

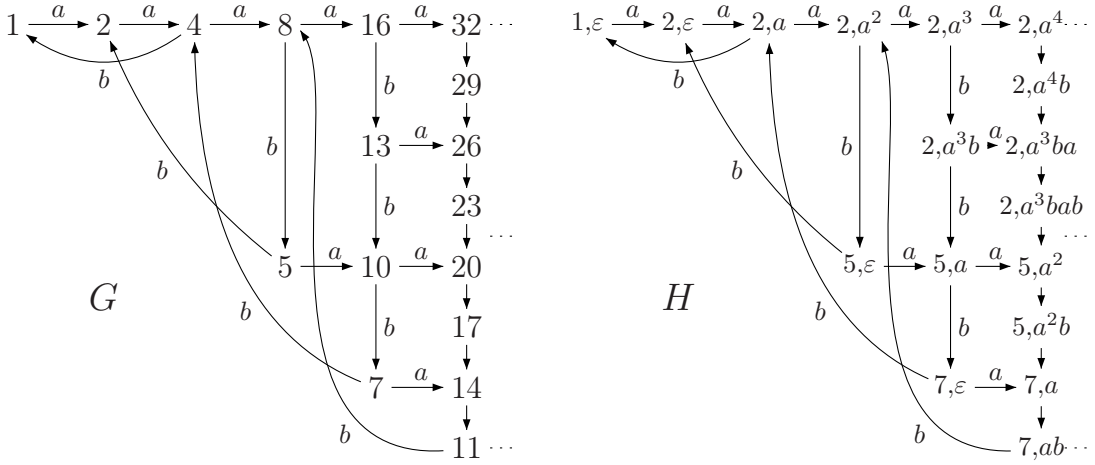


FIG. 4.19 – *Graphe rationnel de degré borné G et graphe linéairement borné isomorphe H.*

---

est une bijection de V_G dans V_H, qui induit un isomorphisme entre G et H. Par conséquent, G est linéairement borné, ce qui conclut la démonstration. □

Exemple 4.73. Soit Γ un alphabet à un seul symbole, et considérons les deux transductions $T_a = \{(n,2n) | n \geq 1\}$ et $T_b = \{(n,n-3) | n \geq 4\}$ sur le domaine des mots sur Γ vus comme des entiers positifs codées en base 1. Appliquons la construction précédente au graphe rationnel de degré borné G défini par T_a et T_b. Ici $<_\Gamma$ est trivial, et nous posons $a <_X b$. Le graphe G et le graphe linéairement borné H obtenu par l'application de la construction sont représentés sur la figure 4.19. Notons par exemple comment le sommet 13 de G est représenté par le sommet $(2,a^3b)$ dans H parce que $(2,a^3b)$ est la plus petite paire (u,v) telle que $u \xrightarrow[G]{v} 13$.

En réalité, on peut démontrer que l'inclusion précédente est stricte.

Théoreme 4.74. *Il existe un graphe linéairement borné de degré borné qui n'est pas un graphe rationnel.*

Démonstration. Commençons par établir le fait que les graphes linéairement bornés de degré borné ne sont pas clos par renversement des arcs. Soit $L \subseteq \{0,1\}^+$ un langage dans NSPACE$[2^n] \setminus$ NSPACE$[n]$ (cf. théorème 1.12), on définit un graphe linéairement borné G de sommets $0\{0,1\}^+\#^* \cup \bar{0}\{\bar{0},\bar{1}\}^+$ et d'arcs définis par

$$\xrightarrow{x} = \{(w,wx) \mid w \in 0\{0,1\}^*\} \cup \{(w,w\bar{x}) \mid w \in \bar{0}\{\bar{0},\bar{1}\}^*\} \qquad \text{pour } x \in \{0,1\},$$

$$\xrightarrow{\#} = \{(w,w\#) \mid w = uv, u \in 0\{0,1\}^*, v \in \#^* \text{ et } |v| < 2^{|u|}\},$$
$$\cup \{(w,\bar{u}) \mid w = uv, u \in 0\{0,1\}^*, v \in \#^*, |v| = 2^{|u|} \text{ et } u \in L\}.$$

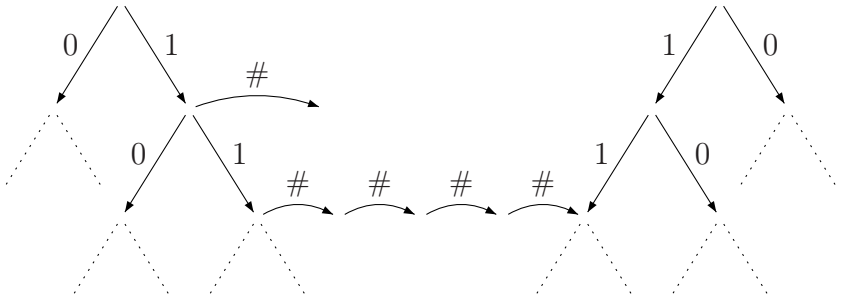


FIG. 4.20 – *Le graphe G pour un certain L contenant* 11.

Les relations $\xrightarrow{0}$, $\xrightarrow{1}$ et $\xrightarrow{\#}$ sont des transductions contextuelles incrémentales. En fait, comme L appartient à NSPACE[2^n], le langage $\{w\#^{2^{|w|}} \mid w \in L\}$ est dans NSPACE[n]. La construction de G est illustrée à la figure 4.20.

Supposons que les graphes linéairement bornés de degré borné soient clos par renversement d'arcs. Il s'ensuivrait logiquement que le graphe H obtenu à partir de G en renversant les arcs étiquetés par # soit aussi un graphe linéairement borné de degré borné. Supposons que ce soit le cas, on aurait que $\xrightarrow[H]{0}$, $\xrightarrow[H]{1}$ et $\xrightarrow[H]{\#}$ sont des transductions contextuelles incrémentales. Soit x le sommet de H correspondant à $\bar{0}$. L'ensemble de sommets $F = \mathrm{Dom}(\xrightarrow[H]{\#})$ est un langage contextuel. Donc d'après la proposition 4.54, $L(H,\{x\},F)$ doit être contextuel. $L = L(H,\{x\},F) \cap \{0,1\}^*$, ceci impliquerait que L soit aussi contextuel, ce qui contredit sa définition.

Comme les graphes rationnels de degré borné sont fermés par inversion d'arcs, on peut déduire du théorème 4.72 qu'ils sont strictement inclus dans les graphes linéairement bornés de degré borné. □

Il peut être intéressant à ce point de notre présentation de rappeler qu'il existe des raisons assez fortes de penser que les langages acceptés par les graphes synchronisés de degré fini sont strictement inclus dans les langages contextuels (cf. théorème 4.32). De plus, toutes les démonstrations existantes que les graphes rationnels acceptent tous les langages contextuels échouent lorsque le degré sortant est borné. Il est donc difficile de dire si les graphes rationnels de degré borné acceptent tous les langages contextuels ou non. Cependant, comme noté à la proposition 4.60, c'est encore le cas des graphes linéairement bornés de degré borné, et aussi en particulier le cas des graphes linéairement bornés déterministes.

4.5 Une autre hiérarchie de graphes à la Chomsky

Nos observations structurelles concernant le degré et le nombre de sommets initiaux des graphes infinis vus comme des automates nous amènent à considérer une notion plus restreinte d'automates infinis que celle apportée par la hiérarchie de familles de graphes proposée au début de ce chapitre (cf. § 4.1), une notion plus proche des automates finis classiques (comme nous l'avons déjà observé dans [Car05]), et nous amènent à proposer une hiérarchie de familles de graphes infinis de degré borné à sommet initial unique acceptant les familles de la hiérarchie de Chomsky, dans l'idée de [CK02a].

Les graphes finis ont bien sûr un degré borné, et acceptent les langages rationnels. Les graphes de transition d'automates à pile temps-réel, qui acceptent tous les langages hors-contexte, sont les graphes de Muller et Schupp [MS85], ou de façon équivalente les graphes HR-équationnels de degré borné [Cou89] et les graphes préfixe-reconnaissables de degré borné [CK01] (suivant qu'une restriction implicite par accessibilité est considérée ou pas). D'après la proposition 4.60, les langages des arbres linéairement bornés déterministes, qui sont bien entendu de degré borné, acceptent les langages contextuels (c'est donc aussi le cas des graphes linéairement bornés en général). Remarquons que, même si nous savons que les graphes rationnels de degré borné sont une sous-famille stricte de ces graphes (théorème 4.72), nous ne savons pas s'ils acceptent tous les langages contextuels. Enfin, puisque les machines de Turing déterministes ont des graphes de transition de degré borné (ou peuvent en avoir) et acceptent tous les langages récursivement énumérables, on peut fermer cette hiérarchie par la famille des graphes de Turing de degré borné. Toutes ces familles sont reprises sur la figure 4.21.

Remarquons que toutes ces familles de graphes sont caractérisées comme des graphes de transition de machines, et que les familles de machines correspondantes forment elles aussi une hiérarchie (syntaxique) stricte.

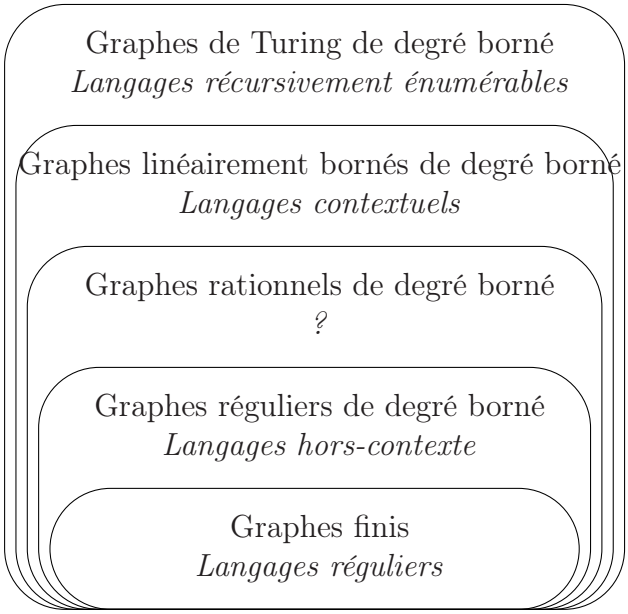


FIG. 4.21 – *Une hiérarchie à la Chomsky de graphes infinis de degré borné.*

Chapitre 5

Analyse d'accessibilité de processus hors-contexte d'ordre supérieur

Les automates et systèmes à pile d'ordre supérieur sont un formalisme très puissant pour la modélisation de programmes fonctionnels d'ordre supérieur, c'est à dire des programmes dont les fonctions et procédures peuvent accepter comme paramètres ou renvoyer comme résultats d'autres fonctions. Par exemple, une traduction des schémas de programmes d'ordre supérieur en automates d'ordre supérieur, et réciproquement, peut être trouvée dans [KNU02].

Nous savons d'autre part que des résultats récents dans le domaine des graphes infinis ont établi que la théorie du second ordre monadique des graphes de transition de ces automates est décidable (cf. § 2.2.1.5). Il reste néanmoins pour que de réelles applications de ces résultats voient le jour à mettre au point des algorithmes ainsi que des méthodes adaptées pour traiter ces modèles.

Dans ce chapitre, nous nous intéressons à la possibilité d'étendre aux systèmes à pile d'ordre supérieur d'un algorithme symbolique d'analyse d'accessibilité présenté dans [BEM97]. Nous proposons une réponse partielle pour une sous-famille de ces systèmes appelés processus hors-contexte d'ordre supérieur. Les représentations symbolique choisies pour des ensembles infinis de configurations sont les langages réguliers. Nous présentons dans un premier temps le modèle étudié ainsi que la représentation symbolique d'ensembles de piles d'ordre supérieur. Enfin, nous détaillons étape par étape notre algorithme d'analyse d'accessibilité.

5.1 Processus hors-contexte d'ordre supérieur

Nous introduisons une classe de modèles appelés *processus hors-contexte d'ordre supérieur*, qui généralisent les processus hors-contexte (CFP) et sont une sous-famille des automates à pile d'ordre supérieur (cf. § 1.3.4). Dans un premier temps, nous définissons un nouvel ensemble d'opérations de *réécriture de niveau*

1 $\{push_1^w \mid w \in \Gamma^*\}$ sur les piles d'ordre supérieur, chacune étant équivalente à une opération pop_1 suivie d'une séquence d'opérations $push_1$. Formellement, soit $w = a_1 \dots a_k \in \Gamma^*$ une séquence de symboles de pile, nous définissons $push_1^w$ comme $pop_1 \circ push_1^{a_k} \circ \dots \circ push_1^{a_1}$. Remarquons en particulier que $push_1^\varepsilon = pop_1$ et que, lorsque le symbole de sommet de pile d'une pile s est a, $push_1^b a(s) = push_1^b(s)$.

À l'aide de ces nouvelles opérations, nous définissons maintenant une sous-famille des automates à pile d'ordre supérieur, qui coïncide au niveau 1 avec la classe déjà connue des *processus hors-contexte* (CFP)[1] [BK89].

Définition 5.1. *Un* processus hors-contexte d'ordre supérieur *de niveau n (ou n-HCFP) est un couple* $H = (\Gamma,\delta)$*, où* Γ *est un alphabet fini et* δ *un ensemble fini de transitions de la forme* (a,o)*, avec* $a \in \Gamma$ *et*

$$o \in \{push_1^w \mid w \in \Gamma^*\} \cup \{push_n, pop_n \mid n > 1\}.$$

Comme nous ne nous intéressons pas ici aux langages acceptés par cette classe d'automates, mais plutôt par la relation qu'ils définissent entre leurs configurations, nous ne considérons que des règles de transition non étiquetées. Une configuration d'un n-HCFP H est une pile de niveau n sur Γ. H définit une relation de transition $\xrightarrow[H]{}$ entre piles de niveau n (ou $\longrightarrow$ si H est compris), avec

$$s \xrightarrow[H]{} s' \iff \exists (a,o) \in \delta \text{ tel que } top(s) = a \text{ et } s' = o(s).$$

Comme les règles de type $push_1$ sont définies à l'aide des opérations classiques des automates à pile d'ordre supérieur, les processus hors-contexte d'ordre supérieur peuvent être vus comme des automates à pile d'ordre supérieur possédant un état de contrôle unique et dont les opérations de niveau 1 peuvent être enchaînées, de la même façon que les processus hors-contexte sont des automates à pile à un état de contrôle.

Le niveau $\mathrm{l}(d)$ d'une transition $d = (a,o)$ est simplement le niveau de o. Définissons quelques notations supplémentaires concernant les calculs de HCFP. Soit $H = (\Gamma,\delta)$ un n-HCFP. Un *calcul* de H à partir d'une pile s_0 est une séquence $s_0 s_1 \dots s_k$ telle que pour tout $i \in [1,k]$, $s_{i-1} \longrightarrow s_i$. La fermeture réflexive et transitive de $\longrightarrow$ est notée $\xrightarrow{*}$ et appelée relation d'accessibilité de H. Pour un ensemble C donné de piles de niveau n, on définit aussi la relation de *transition contrainte* $\longrightarrow_C = \longrightarrow \cap (C \times C)$, et sa fermeture réflexive et transitive $\xrightarrow{*}_C$. Pour tout ensemble S de piles de niveau n, on définit les ensembles :

$$post_H[C](S) = \{\, s \mid \exists s' \in S,\ s' \longrightarrow_C s \,\},$$
$$post_H^*[C](S) = \{\, s \mid \exists s' \in S,\ s' \xrightarrow{*}_C s \,\},$$
$$pre_H[C](S) = \{\, s \mid \exists s' \in S,\ s \longrightarrow_C s' \,\},$$
$$pre_H^*[C](S) = \{\, s \mid \exists s' \in S,\ s \xrightarrow{*}_C s' \,\}.$$

1. Aussi appelés *basic process algebras* dans le domaine des algèbres de processus.

Quand C est l'ensemble $\S_n$ de toutes les piles de niveau n, on l'omet dans la notation et on écrit simplement par exemple $pre_H(S)$ au lieu de $pre_H[C](S)$. On omettra aussi H quand il est clair d'après le contexte. Si H ne possède qu'une seule transition d, on écrira aussi $pre_d(S)$ plutôt que $pre_H(S)$.

5.2 Ensembles de piles et représentation symbolique

Pour être en mesure de mettre au point des techniques de vérification symboliques sur les processus hors-contexte d'ordre supérieur, nous avons besoin d'un moyen de représenter de façon finie des ensembles potentiellement infinis de configurations. Dans ce paragraphe nous présentons la représentation symbolique choisie pour les ensembles de configurations (c'est à dire de piles), ainsi que la famille d'automates qui y est associée.

Une n-pile $s = [s_1 \dots s_l]$ sur Γ est associée à un mot $w(s) = [w(s_1) \dots w(s_l)]$, dans lequel les lettres de Γ n'apparaissent qu'à une profondeur de parenthésage égale à n (nous omettrons souvent de distinguer une pile s du mot $w(s)$ associé). Un ensemble de piles sur Γ est dit *régulier* si l'ensemble des mots le représentant est accepté par un automate fini sur $\Gamma' = \Gamma \cup \{\,[\,,\,]\,\}$, que nous appelons dans ce cas précis un *automate de piles*.

Soit A un tel automate, un état p de A est dit de niveau 0 s'il n'a aucune transition sortante par [et aucune transition entrante par]. Il est de niveau k si tous ses successeurs par [et prédécesseurs par] sont de niveau $k-1$. Le niveau de p est noté $\mathrm{l}(p)$. Nous définissons également une notion de niveau pour les chemins. Un *chemin de niveau n* est un chemin $p_1 \dots p_k$ tel que $\mathrm{l}(p_1) = \mathrm{l}(p_k) = n$ et $\forall i \in [2,k-1]$, $\mathrm{l}(p_i) < n$. Tous les chemins de niveau n d'un automate sont étiquetés par des piles de niveau n. Enfin, pour pouvoir faire référence de manière concise à l'ensemble des chemins de niveau n entre deux états de niveau n donnés, on introduit la notation suivante. Soit

$$Q = \{\, q \in Q_A \mid \mathrm{l}(q) < n \wedge p_1 \longleftrightarrow A{+}q \longleftrightarrow A{+}p_2 \,\}$$

l'ensemble de tous les états de A apparaissant le long d'un chemin de niveau n entre p_1 et p_2. Si Q est non vide, on écrit $p_1 \overset{B}{\underset{A}{\rightsquigarrow}} p_2$, où B est défini comme

$$B = \big(\, Q_B = Q \cup \{p_1,p_2\},\, \Gamma',\, \delta_B = \delta_A \cap (Q_B \times \Gamma' \times Q_B),\, p_1,\, p_2 \,\big).$$

Cette notation permet en quelque sorte d'isoler des « sous-automates » de niveau inférieur au sein d'un automate de niveau plus élevé. Si l'on suit cette idée, et en raison de la structure imbriquée des piles, il sera parfois plus commode

de caractériser directement des ensembles de piles à l'aide d'*automates de piles hiérarchiques*.

Définition 5.2. *Un automate de piles hiérarchique de niveau 1 est un automate fini étiqueté sur l'alphabet de pile* Γ. *Un automate de pile hiérarchique de niveau* $n \geq 2$ *est un automate fini dont les transitions sont étiquetées par des automates hiérarchiques de niveau* $n-1$ *sur* Γ.

Soit $A = (Q,\Gamma,\delta,q_0,q_f)$ un automate hiérarchique de niveau n[2] avec $n \geq 2$. L'existence d'une transition étiquetée par un automate B de niveau 1 entre les états de contrôle p et q dans A est notée comme de coutume $p \longleftrightarrow ABq$, ou simplement $p \xrightarrow{B} q$. Le langage de niveau k de A, pour $k \in [1,n]$, est défini récursivement comme

$$\begin{aligned} L_k(A) &= \{\, [L_k(A_1) \dots L_k(A_l)] \mid [A_1 \dots A_l] \in L_n(A) \,\} && \text{si } k < n, \\ L_k(A) &= \{\, [A_1 \dots A_l] \mid q_0 \longleftrightarrow AA_1 \dots \longleftrightarrow AA_l q_f \,\} && \text{si } k = n. \end{aligned}$$

Par simplicité, on abrège souvent $L_1(A)$ en $L(A)$. On dit qu'un automate hiérarchique B apparaît dans A si B étiquette une transition de A, ou apparaît au sein d'un automate étiquetant une transition de A.

Les automates hiérarchiques sont utiles pour représenter des ensembles de piles d'ordre supérieur, mais on peut montrer qu'ils ont en fait la même expressivité que les automates de piles ordinaires.

Proposition 5.1. *Les langages de piles acceptés par les automates hiérarchiques de piles sont les langages réguliers de piles.*

Démonstration. Soit $A = (Q,\Gamma,\delta,i,f)$ un automate hiérarchique, nous calculons par récurrence sur le niveau n de A un automate de piles non hiérarchique $\flat A$ tel que $L_1(A) = L(\flat A)$. Pour $n = 1$, on prend $\flat A = A$. Pour les valeurs plus élevées de n, soient $A_1 \dots A_m$ les automates hiérarchiques de niveau $n-1$ étiquetant les transitions de A. Par hypothèse de récurrence on peut construire des automates $\flat A_1 \dots \flat A_m$ tels que $\forall j \in [1,m]$, $L_1(A_j) = L(\flat A_j)$. Soit $\flat A_j = (Q_j,\Gamma,\delta_j,i_j,f_j)$, (tous les Q_j sont supposés disjoints). Nous construisons alors l'automate $\flat A = (Q',\Gamma,\delta',i',f')$ où pour tous $p,q \in Q$, $j \in [1,m]$, $r,s,t,u \in Q_j$ et $a \in \Gamma'$ tels que $p \longleftrightarrow AA_jq$, $i_j \longleftrightarrow A_j\downarrow[r$, $s \longleftrightarrow A_j\downarrow at$ et $u \longleftrightarrow A_j\downarrow]f_j$, on a :

$$i' \longleftrightarrow A\downarrow[i \quad p \longleftrightarrow A\downarrow[pr \quad ps \longleftrightarrow A\downarrow apt \quad pu \longleftrightarrow A\downarrow]q \quad f \longleftrightarrow A\downarrow]f'.$$

Suivant cette construction, un chemin de $\flat A$ entre deux états de contrôle p et q dans $Q \cap Q'$ est étiqueté par le mot s si et seulement si s représente une pile de niveau $(n-1)$ acceptée par un certain A_j tel que $p \longleftrightarrow AA_jq$. Ainsi $\flat A$ accepte tous les mots de la forme $[s_1 \dots s_l]$ tels que $[A_{i_1} \dots A_{i_l}] \in L_n(A)$ et tel que pour tout j, $s_j \in L(A_{i_j})$, ce qui est précisément la définition de $L(A)$.

2. Notez que nous considérons des automates à état final unique.

Réciproquement, soit $A = (Q,\Gamma',\delta,i,f)$ un automate acceptant des piles de niveau n. Nous voulons construire un automate hiérarchique $A' = (Q',\Gamma,\delta',i',f')$ de niveau n tel que $L_1(A') = L(A)$. Comme aucun chemin de A étiqueté par un mot qui ne représente pas une pile ne peut être acceptant, on peut supposer que le niveau de chaque état dans Q est bien défini. Soit Q_{n-1} l'ensemble d'états de niveau $n-1$ de A. Les seuls états de niveau n sont i et f. Si $n = 1$, on construit A' avec un ensemble d'états $Q' = Q_{n-1}$ et l'ensemble de transitions

$$\delta' = \{\, i' \xrightarrow{a} q \mid i \longleftrightarrow A[p \longleftrightarrow Aaq \,\} \;\cup\; \big(\delta \cap (Q_{n-1} \times \Gamma \times Q_{n-1})\big) \\ \cup\; \{\, p \xrightarrow{a} f' \mid p \longleftrightarrow Aaq \longleftrightarrow A]f \,\}.$$

Si $n > 1$, pour tous $p,q \in Q_{n-1}$ et tout automate B tel que $p \overset{B}{\rightsquigarrow} q$, on construit d'abord récursivement l'automate hiérarchique B' de niveau $n-1$ tel que $L_1(B') = L(B)$. On donne ensuite à A' l'ensemble de transitions

$$\delta' = \{\, i' \xrightarrow{B'} q \mid \exists p,q \in Q_{n-1},\, i \longleftrightarrow A[p \overset{B}{\underset{A}{\rightsquigarrow}} q \,\} \\ \cup\; \{\, p \xrightarrow{B'} q \mid \exists p,q \in Q_{n-1},\, p \overset{B}{\underset{A}{\rightsquigarrow}} q \,\} \\ \cup\; \{\, p \xrightarrow{a} f' \mid \exists p,q \in Q_{n-1},\, p \overset{a}{\underset{A}{\rightsquigarrow}} q \longleftrightarrow A]f \,\}.$$

Une pile s est acceptée par A' si et seulement si il existe un chemin dans A' étiqueté par $B'_1 \dots B'_k$ de i' à f' tel que $s \in [L_1(B'_1) \dots L_1(B'_k)]$. Nous avons donc également $s \in [L(B_1) \dots L(B_k)]$, et par conséquent $s \in L(A)$. □

De plus, l'ensemble des langages réguliers de piles de niveau n est fermés par union, intersection et complémentaire dans $\S_n$. Pour tout $a \in \Gamma$ et $n \in \mathbb{N}$, on définit pour un usage ultérieur l'automate A^n_a reconnaissant le langage $L(A^n_a) = \{\, s \in \S_n \mid top(s) = a \,\}$. On note aussi $A \times B$ l'opération de produit synchrone classique entre deux automates tel que $L(A \times B) = L(A) \cap L(B)$.

Avant de continuer, nous présentons quelques définitions et notations supplémentaires. Pour pouvoir facilement exprimer l'existence d'ensembles de calculs d'automates hiérarchiques et les manipuler aisément, nous définissons la notion d'*expression de pile*.

Définition 5.3. *Une expression de pile de niveau* 0 *sur l'alphabet* Γ *est simplement une lettre dans* Γ. *Une expression de pile de niveau* $n > 0$ *est soit une pile* s *de niveau* n, *le nom* A *d'un automate de piles de niveau* n *(hiérarchique ou non), une concaténation d'expressions de piles de niveau* n, *une expression de pile de niveau* $n-1$ *entre crochets droits* $[e]$, *ou la concaténation itérée* e^+ *d'une expression de pile* e *de niveau* n.

De plus, pour décrire les calculs d'automates hiérarchiques nous définissons une relation binaire $\longmapsto$, qui exprime le choix d'un chemin particulier dans un automate hiérarchique apparaissant au sein d'une expression.

Définition 5.4. *Soit $e = uAv$ une expression de piles où A est un automate hiérarchique de piles de niveau n, on peut écrire $e \longmapsto u[w]v$ pour tout $w \in L_n(A)$. Comme d'habitude on note $\overset{*}{\longmapsto}$ la clôture réflexive et transitive de $\longmapsto$. Une séquence d'expressions de piles $e_1 \dots e_m$ telles que $e_1 = A$, $e_m \in \S_n$ et $\forall i \in [1,m-1]$, $e_i \longmapsto e_{i+1}$ est appelée un calcul hiérarchique de A.*

Enfin, nous définissons une notion de concaténation sur les piles et expressions de piles.

Définition 5.5. *Soient $e = [e_1 e_2]$, f et g des expressions de piles, on écrit $e = f \cdot g$ si soit $f = e_1$ et $g = [e_2]$, soit $e_1 = f \cdot g'$ et $g = [g' e_2]$. Remarquons que si e est une lettre de Γ ou un automate, il n'existe aucun f et g tels que $e = f \cdot g$.*

Par exemple, on pourrait écrire

$$[[aB][a][bcd]] = a \cdot [[B][a][bcd]], \text{oubien} [[aB][a][bcd]] = [aB][a] \cdot [[bcd]].$$

5.3 Analyse symbolique d'accessibilité

Notre but dans ce paragraphe est de chercher à mettre au point des techniques effectives de calcul des ensembles $pre(S)$, $post(S)$, $pre^*(S)$ et $post^*(S)$ pour un n-HCFP H donné, dans le cas où S est un langage régulier de piles. Pour les automates à pile de niveau 1, il est bien connu que $pre^*_H(S)$ et $post^*_H(S)$ sont réguliers, et plusieurs algorithmes existent pour les calculer. Nous allons voir que c'est toujours le cas pour $pre(S)$ et $pre^*(S)$ dans le cas des HCFP, mais pas pour $post(S)$ (et donc non plus pour $post^*(S)$).

5.3.1 Accessibilité vers l'avant

Proposition 5.2. *Étant donné un n-HCFP H et un ensemble régulier S de piles de niveau n, l'ensemble $post(S)$ est en général non régulier.*

Démonstration. Soit $post_{(a,o)}(S)$ l'ensemble $\{ s' \mid \exists s \in S,\ top(s) = a \ \wedge \ s' = o(s) \}$. Supposons que S est un langage régulier de piles de niveau n, alors si $d = (a,push^w_1)$ ou $d = (a,pop_k)$, il est facile de voir que $post_{(a,o)}(S)$ est régulier. Cependant, si $d = (a,push_k)$ avec $k > 1$, alors $post_{(a,o)}(S)$ est l'ensemble $\{ [^{n-k+1} t\, t\, w \mid [^{n-k+1} t\, w \in S \}$. Il est clair que cet ensemble n'est pas régulier en raison de la duplication de t (il s'agit en fait un langage contextuel). □

5.3.2 Accessibilité vers l'arrière

Nous proposons tout d'abord une transformation sur les automates qui correspondent à l'opération *pre* appliquée à leur langage. Dans un second temps, nous étendons cette construction au calcul plus délicat des ensembles pre^*.

Proposition 5.3. *Étant donnés un n-HCFP H et un ensemble régulier S de piles de niveau n, l'ensemble $pre(S)$ est régulier et effectivement calculable.*

Nous introduisons une construction qui, pour une transition de HCFP d donnée et un ensemble régulier S de piles reconnu par un automate hiérarchique A de niveau n, nous permet de calculer un automate hiérarchique A'_d dont le langage est l'ensemble $pre(S)$ des prédécesseurs directs de S par d. Cette construction est une transformation portant sur les automates hiérarchiques, que nous appelons T_d. On définit $A'_d = T_d(A) = (Q',\Gamma,\delta',q'_0,q_f)$ comme suit.
Si $\mathrm{l}(d) < n$, on propage la transformation au premier automate de niveau $n-1$ rencontré le long de chaque chemin de A. On a donc $Q' = Q$, $q'_0 = q_0$ et

$$\delta' = \{\, q_0 \xrightarrow{T_d(A_1)} q_1 \mid q_0 \longleftrightarrow AA_1q_1 \,\} \cup \{\, q \xrightarrow{B} q' \mid q \longleftrightarrow ABq' \wedge q \neq q_0 \,\}.$$

Si $\mathrm{l}(d) = n$, on distingue trois cas suivant la nature de d:

1. Si $d = (a,push_1^w)$, alors $Q' = Q \cup \{q'_0\}$ et $\delta' = \delta \cup \{\, q'_0 \xrightarrow{a} q_1 \mid q_0 \longleftrightarrow Awq_1 \,\}$.
2. Si $d = (a,push_n)$ et $n > 1$, alors $Q' = Q \cup \{q'_0\}$ et $\delta' = \delta \cup \{\, q'_0 \xrightarrow{B} q_2 \mid \exists q_1,\ q_0 \longleftrightarrow AA_1q_1 \longleftrightarrow AA_2q_2 \,\}$ avec $B = A_1 \times A_2 \times A_a^{(n-1)}$.
3. Si $d = (a,pop_n)$, alors $Q' = Q \cup \{q'_0\}$ et $\delta' = \delta \cup \{\, q'_0 \xrightarrow{A_a^{(n-1)}} q_0 \,\}$.

Il n'est pas difficile de démontrer que $L(A'_d) = pre_d(L(A))$. Donc, si γ est l'ensemble des règles de transition de H, on a $pre(S) = pre(L(A)) = \bigcup_{d\in\gamma} L(A'_d)$.

Cette technique peut être étendue afin de calculer l'ensemble $pre^*(S)$ de tous les prédécesseurs d'un ensemble de piles S.

Théoreme 5.4. *Étant donnés un n-HCFP H et un ensemble régulier S de piles de niveau n, l'ensemble $pre^*(S)$ est régulier et effectivement calculable.*

Pour calculer $pre^*(S)$, il est nécessaire de régler le problème de la terminaison. Une itération naïve de la transformation T_d précédente ne termine en général pas, car chaque étape ajoute de nouveaux états de contrôle à l'automate. En fait, même la simple séquence $(pre^i(S))_{i\geq 0}$, définie comme $pre^0(S) = S$ et, pour tout $n \geq 1$, $pre^n(S) = pre^{n-1}(S) \cup pre(pre^{n-1}(S))$, n'atteint en général jamais un point fixe. Par exemple, si $d = (a,pop_1)$, alors pour tout n, $pre^n([a]) = \{\, [a^i] \mid i \leq n \,\} \neq pre^{n+1}([a])$.

Pour construire $pre^*(S)$ pour un ensemble régulier S, nous modifions la construction précédente afin de garder borné le nombre des états de l'automate hiérarchique que nous manipulons. L'idée, au lieu de créer de nouveaux états de

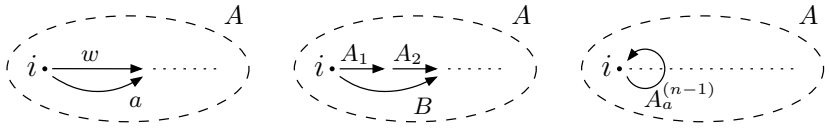


FIG. 5.1 – *La transformation $T_d(A)$ pour $d = (a,push_1^w)$, $(a,push_k)$ et (a,pop_k).*

contrôle, est d'ajouter des arcs à l'automate jusqu'à la saturation, en créant si nécessaire des circuits dans l'automate pour représenter en une fois un nombre quelconques d'applications répétées d'une certaine séquence d'opérations. Ensuite, nous démontrons que ce nouvel algorithme termine et est correct.

Définissons tout d'abord l'opération T_d pour toute transition d de n-HCFP (voir la figure 5.1 pour une illustration). Soient $A = (Q,\Gamma,\delta,q_0,q_f)$ et $A' = (Q,\Gamma,\delta',q_0,q_f)$ deux automates hiérarchiques de niveau n sur $\Gamma' = \Gamma \cup \{ [,] \}$, et d une transition de n-HCFP. Nous définissons $A' = T_d(A)$ de la façon suivante.
Si le niveau de d est inférieur à n, on propage simplement la transformation au premier automate de niveau $n-1$ rencontré le long de chaque chemin :

$$\delta' = \{ q_0 \xrightarrow{T_d(A_1)} q_1 \mid q_0 \longleftrightarrow AA_1q_1 \} \cup \{ q \xrightarrow{B} q' \mid q \longleftrightarrow ABq' \wedge q \neq q_0 \}.$$

Si $\mathrm{l}(d) = n$ alors comme précédemment on distingue trois cas en fonction de d:

1. Si $n = 1$ et $d = (a,push_1^w)$, alors $\delta' = \delta \cup \{ q_0 \xrightarrow{a} q_1 \mid q_0 \longleftrightarrow Awq_1 \}$.
2. Si $d = (a,push_n)$ pour $n > 1$, alors
   $\delta' = \delta \cup \{ q_0 \xrightarrow{B} q_2 \mid \exists q_1,\ q_0 \longleftrightarrow AA_1q_1 \longleftrightarrow AA_2q_2 \}$ avec $B = A_1 \times A_2 \times A_a^{(n-1)}$.
3. Si $d = (a,pop_n)$, alors $\delta' = \delta \cup \{ q_0 \xrightarrow{A_a^{(n-1)}} q_0 \}$

Supposons que $H = (\Gamma,\gamma)$ avec $\gamma = \{ d_0,\ldots,d_{l-1} \}$. Étant donné un automate A tel que $S = L(A)$, considérons la séquence $(A_i)_{i\geq 0}$ définie par $A_0 = A$ et pour tout $i \geq 0$ et $j = i \mod l$, $A_{i+1} = T_{d_j}(A_i)$. Afin d'obtenir le résultat recherché, nous devons démontrer que cette séquence finit toujours par atteindre un point fixe (Lemme 5.5) et que ce point fixe est un automate reconnaissant effectivement $pre^*(S)$ (Lemmes 5.8 et 5.9).

5.3.2.1 Terminaison et complexité

Lemme 5.5 (Terminaison)**.** *Pour tout automate A de piles de niveau n et tout n-HCFP $H = (\Gamma,\delta)$, la séquence $(A_i)_{i\geq 0}$ définie par rapport à A se stabilise en un nombre fini d'étapes: $\exists k \geq 0$, $\forall k' \in \delta$, $A_{k'} = A_k$, ce qui implique que $L(A_k) = \bigcup_{i\geq 0} L(A_i)$.*

Démonstration. Tout d'abord, remarquons que pour tout d, T_d ne modifie l'ensemble d'états de contrôle d'aucun automate apparaissant dans A, mais ne fait

que leur ajouter des transitions. Cela signifie que la taille des automates de la séquence $(A_i)_{i\geq 0}$ croît avec i.

Pour établir la terminaison de la construction, nous démontrons que le nombre de transitions qui peuvent être ajoutées à l'automate initial A_0 est fini. Remarquons que par définition de T_d, le nombre d'états de chaque A_i est constant. De plus, chaque nouvelle transition a pour origine l'état initial de l'automate dans laquelle elle est ajoutée. Par conséquent, le nombre total de transitions qui peuvent être ajoutées à un automate donné est égal à $|V_n| \cdot |Q|$, où V_n est le vocabulaire de niveau n et Q l'ensemble d'états de l'automate. Comme $|Q|$ ne change pas, il suffit de prouver que V_n est fini pour tout n. Si $n = 1$, $V_1 = \Gamma$, et la propriété est vraie. Supposons que la propriété est vraie jusqu'à un certain rang $n-1 > 0$. Par hypothèse de récurrence, V_{n-1} est fini. Avec cet ensemble d'étiquettes, on peut construire un nombre N d'automates de niveau $n-1$ qui est exponentiel en $|V_{n-1}| \cdot K$, où K dépend du nombre d'automates de niveau $n-1$ dans A_0 et de leurs ensembles d'états de contrôle. Comme chaque transition d'un automate de niveau n est étiquetée par un produit d'automates de niveau $n-1$, $|V_n|$ est lui-même exponentiel en N, et donc doublement exponentiel en $|V_{n-1}|$. Par conséquent, la complexité totale de l'algorithme est non-élémentaire en n. □

5.3.2.2 Consistance

Le lemme élémentaire suivant exprime le simple fait que si une transition (a,pop_k) peut être appliquée à une certaine pile, alors elle doit aussi pouvoir s'appliquer à toute pile de même niveau et de même sommet de pile.

Lemme 5.6. *Pour tout HCFP H et expression de pile constante*[3] *s,*

$$\exists t,\ s \cdot t \xrightarrow{*} t \implies \forall t',\ s \cdot t' \xrightarrow{*} t'.$$

Démonstration. La démonstration se fait par un simple raisonnement par récurrence sur la taille de l'expression s. □

Avant de démontrer la consistance de la construction de la proposition 5.4, nous énonçons un lemme technique exprimant le fait que tout circuit sur l'état initial d'un automate hiérarchique au cours du calcul de la séquence (A_i) correspond à des calculs valides du processus considéré.

Lemme 5.7. *Pour tout $i \geq 0$ et tout automate hiérarchique de piles de niveau k $B = (Q,\Gamma,\delta,q_0,q_f)$ apparaissant dans A_i, s'il existe un état $q_1 \neq q_0$, des étiquettes de chemins w_1 et w_2, une étiquette de transition C et un chemin $q_0 \xrightarrow{w_1} q_0 \xrightarrow{C} q_1 \xrightarrow{w_2} q_f$ dans B, alors pour tout calcul*

$$A_i \xmapsto{*} B \cdot r \xmapsto{*} [\, w_1\, C\, w_2\,] \cdot r \xmapsto{*} t \cdot s$$

3. On dit qu'une expression est constante si elle ne contient pas de nom d'automate.

où r est une expression de pile quelconque, $w_1 \overset{*}{\longmapsto} t$ et $[\,C\,w_2\,]\cdot r \overset{*}{\longmapsto} s$, on a nécessairement $s \in pre^*_H(t\cdot s)$ et

$$A_i \overset{*}{\longmapsto} B\cdot r \overset{*}{\longmapsto} [\,C\,w_2\,]\cdot r \overset{*}{\longmapsto} s.$$

Démonstration. Raisonnons par récurrence sur i. Supposons par simplicité qu'aucune transition ne mène à l'état initial d'aucun automate apparaissant dans A_0. Supposons la propriété vraie jusqu'à un certain rang $i \geq 0$. Soit d l'opération de niveau k telle que $A_{i+1} = T_d(A_i)$. Considérons le calcul ρ suivant de A_{i+1}:

$$A_{i+1} \overset{*}{\longmapsto} B\cdot r \overset{*}{\longmapsto} [\,w_1\,C\,w_2\,]\cdot r \overset{*}{\longmapsto} t\cdot s \text{ with } w_1 \overset{*}{\longmapsto} t.$$

Comme w_1 étiquette une boucle sur l'état initial de B, un autre calcul possible de A_{i+1} est

$$A_{i+1} \overset{*}{\longmapsto} B\cdot r \overset{*}{\longmapsto} [\,C\,w_2\,]\cdot r \overset{*}{\longmapsto} s.$$

Il nous suffit de montrer que $t\cdot s \overset{*}{\longrightarrow} s$ pour conclure la démonstration. À cette fin, nous raisonnons par récurrence sur le nombre m de nouvelles transitions de niveau k dans A_{i+1} (c'est à dire de transitions de A_{i+1} n'apparaissant pas dans A_i) utilisées dans le circuit w_1 sur q_0.

$m = 0$: Comme w_1 ne contient aucune nouvelle transition, ce mot étiquette aussi un circuit dans A_i. Donc si la transition C appartient à A_i, ρ est un chemin de A_i, et la propriété est vraie par récurrence sur i. Si au contraire C est une nouvelle transition, par définition de A_{i+1}, A_i admet le calcul suivant :

$$A_i \overset{*}{\longmapsto} B\cdot r \overset{*}{\longmapsto} [\,w_1\,u\,w_2\,]\cdot r \overset{*}{\longmapsto} t\cdot s' \text{ avec } w_1 \overset{*}{\longmapsto} t \text{ et } [uw_2]\cdot r \overset{*}{\longmapsto} s',$$

où u est égal à ε, $C_1\,C_2$ ou v quand d vaut respectivement (a,pop_k), $(a,push_k)$ ou $(a,push_1^v)$. Par récurrence sur i, ce calcul vérifie la propriété, donc on a

$$A_i \overset{*}{\longmapsto} B\cdot r \overset{*}{\longmapsto} [\,u\,w_2\,]\cdot r \overset{*}{\longmapsto} s' \text{ avec } t\cdot s' \overset{*}{\longrightarrow} s'.$$

D'après le lemme 5.6, ceci implique que $\forall s''$, $t\cdot s'' \overset{*}{\longrightarrow} s''$, et en particulier $t\cdot s \overset{H}{\underset{*}{\hookrightarrow}} s$.

$m \Rightarrow m+1$: Supposons que le circuit w_1 dans B contienne $m+1$ nouvelles transitions. Soit $q_0 \longleftrightarrow Dq_0$ l'une de ces transitions, on a $w_1 = w_1' C w_2'$. Donc B a un chemin

$$q_0 \longleftrightarrow Bw_1'q_0 \longleftrightarrow BDq_0 \longleftrightarrow Bw_1''q_0 \longleftrightarrow BCq_1 \longleftrightarrow Bw_2q_f$$

commençant par un circuit sur q_0 étiqueté par w_1', contenant au plus m nouvelles transitions de A_{i+1}. Supposons que $t = t_1\cdot t_2$ et $w_1' \overset{*}{\longmapsto} t_1$, par hypothèse de récurrence sur m on a :

$$A_{i+1} \overset{*}{\longmapsto} B\cdot r \overset{*}{\longmapsto} [\,D\,w_1''\,C\,w_2\,]\cdot r \overset{*}{\longmapsto} t_2\cdot s$$

et $s \in pre^*_H(t_1 \cdot s)$. Il reste donc à examiner la façon dont la transition D est créée dans A_{i+1}, ce qui dépend du type de d. Comme précédemment, par définition de A_{i+1} il doit exister un calcul de la forme

$$A_i \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, u\, w_1''\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} t_3 \cdot s,$$

où u vaut ε, $D_1\, D_2$ ou v selon que d est respectivement (a,pop_k), $(a,push_k^v)$ ou $(a,push_1^v)$. On peut facilement montrer que t_3 peut être choisi égal à $d(t_2)$. Ce calcul utilise un chemin dans B démarrant avec un circuit sur q_0 étiqueté par $u\, w_1''$ contenant au plus m nouvelles transitions de niveau k :

$$q_0 \longleftrightarrow Buq_0 \longleftrightarrow Bw_1''q_0 \longleftrightarrow BCq_1 \longleftrightarrow Bw_2q_f.$$

Par hypothèse de récurrence sur m, on peut en conclure que

$$A_{i+1} \overset{*}{\longmapsto} [\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} s \text{ et } t_3 \cdot s \in pre^*_H(s).$$

On obtient $t \cdot s \overset{H}{\underset{*}{\hookrightarrow}} t_2 \cdot s \longrightarrow t_3 \cdot s \overset{H}{\underset{*}{\hookrightarrow}} s$, et donc $t \cdot s \overset{H}{\underset{*}{\hookrightarrow}} s$, ce qui conclut la démonstration.

□

Lemme 5.8 (Consistance)**.** $\bigcup_{i\geq 0} L(A_i) \subseteq pre^*_H(S)$.

Démonstration. On preuve par récurrence sur i le résultat équivalent que

$$\forall i,\ L(A_i) \subseteq pre^*_H(S).$$

Considérons pour plus de simplicité que l'automate $A = A_0$ ne contient aucune transition menant vers son état initial.

Le cas de base $i = 0$ est trivial puisque $A_0 = A$ et $L(A) \subseteq pre^*_H(L(A))$. Considérons maintenant une pile s dans $L(A_{i+1})$. Si s est acceptée par A_{i+1} sans utiliser de nouvelle transition, alors elle est aussi acceptée par A_i. donc par récurrence sur i elle doit appartenir à $pre^*_H(S)$. Dans le cas contraire, le calcul acceptant doit être de la forme

$$A_{i+1} \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, w_1\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} s,$$

où le chemin de B qui engendre $w_1\, C\, w_2$ est de la forme

$$q_0 \longleftrightarrow Bw_1q_0 \longleftrightarrow BCq_1 \longleftrightarrow Bw_2q_f,$$

avec $q_1 \neq q_0$. D'après le lemme 5.7 il existe t,s_1 tels que $s = t \cdot s_1$, $t \cdot s_1 \overset{*}{\longrightarrow} s_1$ et

$$A_{i+1} \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} s_1.$$

Notons que par définition de T_d, toutes les nouvelles transitions doivent avoir pour origine les états initiaux des automates apparaissant dans A_{i+1}. Donc, si la transition étiquetée par C dans le calcul précédent n'est pas nouvelle, alors le calcul tout entier appartient déjà à A_i. Par récurrence sur i, il existe $s_2 \in S$ telle que $s_1 \xrightarrow{*} s_2$, donc par transitivité $s \xrightarrow{*} s_2$.
Si la transition étiquetée par C est nouvelle, alors comme $q_1 \neq q_0$ et par définition de T_d, d doit être de la forme $(a,push_k)$ ou $(a,push_1^v)$. Donc par construction de A_{i+1} il existe un calcul

$$A_{i+1} \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, u\, w_2 \,] \cdot r \overset{*}{\longmapsto} s_2,$$

avec u valant soit $C_1\, C_2$ si $k > 1$, soit v si $k = 1$, et s_2 peut être choisie égale à $d(s_1)$. Donc par hypothèse de récurrence sur i; il existe $s_3 \in S$ telle que $s_2 \xrightarrow{*} s_3$, et donc par transitivité $s \xrightarrow{*} s_3$. □

5.3.2.3 Complétude

Lemme 5.9 (Complétude)**.** $pre^*_H(S) \subseteq \bigcup_{i\geq 0} L(A_i)$.

Démonstration. Nous démontrons la propriété suffisante que pour tout automate hiérarchique de piles A et toute transition d'HCFP d, $pre_d(L(A)) \subseteq L(T_d(A))$. Soient deux automates A et A' tels que $A' = T_d(A)$. Considérons une pile $s \in L(A)$, et soit s' toute pile telle que $s' \in pre_{d_j}(s)$. Il existe un chemin ρ de A acceptant s :

$$A \overset{*}{\longmapsto} B \cdot r \longmapsto [\, C_1 \dots C_l] \cdot r \overset{*}{\longmapsto} s.$$

Selon la valeur de d, on doit considérer trois cas :

1. Si $d_j = (a,pop_k)$, alors $s' = t \cdot s$ où t est une pile quelconque de niveau $k-1$ telle que $top(t) = a$, et par définition de T_d le calcul suivant existe :

$$A' \overset{*}{\longmapsto} [\, A_a^{(k-1)} C_1 \dots C_l \,] \cdot r \overset{*}{\longmapsto} s'.$$

2. Si $d_j = (a,push_k)$, $k > 0$, alors $s = tt \cdot r$ et $s' = t \cdot r$ avec $top(t) = a$ et t est à la fois dans $L(C_1)$ et $L(C_2)$. Donc t est également acceptée par l'automate de niveau $k-1$ $C_1 \times C_2 \times A_a^{k-1}$. Donc, par définition de T_d le calcul suivant existe :

$$A' \overset{*}{\longmapsto} [\, C_1 \times C_2 \times A_a^{k-1}\, C_3 \dots C_l \,] \cdot r \overset{*}{\longmapsto} s'.$$

3. Si $d_j = (a,push_1^w)$, alors $s = w \cdot r$ et $s' = a \cdot r$. Cela implique que $C_1 \dots C_l$ sont des automates de niveau 0 (des lettres), et $C_1 \dots C_{|w|} = w$. Par définition de T_d le calcul suivant existe :

$$A' \overset{*}{\longmapsto} [\, a\, C_{|w|+1} \dots C_l \,] \cdot r \overset{*}{\longmapsto} s'.$$

Ceci établit le fait que T_d ajoute au langage L de son argument *au moins* l'ensemble des prédécesseurs directs de L par l'opération d. □

Comme conséquence directe de la proposition 5.3 et du théorème 5.4, nous obtenons un algorithme de model checking symbolique pour la logique $\mathsf{E}(\mathsf{F},\mathsf{X})$ ayant comme prédicats atomiques des ensembles réguliers de piles, c'est à dire le fragment de la logique temporelle CTL pour les modalités EF et EX.

Théoreme 5.10. *Pour tout HCFP H et toute formule close φ de $\mathsf{E}(\mathsf{F},\mathsf{X})$, l'ensemble des configurations de H satisfaisant φest régulier et calculable de façon effective.*

5.4 Accessibilité contrainte

Dans ce paragraphe nous nous intéressons au problème plus général du calcul d'un automate fini acceptant l'ensemble $pre^*_H[C](S)$ pour tout HCFP H et toute paire de langages réguliers de piles C et S. nous fournissons une extension de la construction de la proposition 5.4 nous permettant d'assurer que nous ne prenons en considération que les calculs de H dont toutes les configurations sont dans C. Une fois encore, partant d'un automate de piles A, nous construisons une séquence d'automates dont le point fixe reconnaît le langage $pre^*_H[C](L(A))$.

La principale (et la seule) différence avec la cas précédent est que nous avons besoin de calculer à chaque étape des intersections de langages sans remettre en question l'argument de terminaison de l'algorithme (c'est à dire sans ajouter de nouveaux états à l'automate de départ). Pour cette raison, nous utilisons une classe d'automates *alternants*, que nous appelons automates hiérarchiques *contraints*.

5.4.1 Automates hiérarchiques contraints

Définition 5.6. *Soit B un automate non hiérarchique de piles de niveau m (avec $m \geq n$). Un automate hiérarchique B-contraint A de niveau n est un automate hiérarchique $(Q_A,\Gamma,\delta_A,i_A,f_A)$ possédant des transitions spéciales de la forme $p \longleftrightarrow AC(q,r)$ où $p,q \in Q_A$, r est un état de contrôle de B et C est un automate hiérarchisé B-contraint de niveau $n-1$.*

Le langage d'un automate hiérarchique contraint est défini via une simple adaptation de la construction de la proposition 5.1. Soit un automate hiérarchique $A = (Q,\Gamma,\delta,i,f)$ de niveau n contraint par un automate non hiérarchique $B = (Q_B,\Gamma',\delta_B,i_B,f_B)$ de niveau n[4]. Premièrement, considérons l'automate hiérarchique non contraint $A' = (Q,\Gamma,\delta',i,f)$, avec $\delta' = \{\, p \xrightarrow{C} q \mid p \longleftrightarrow AC(q,r) \,\}$.

4. Il est important que les niveaux de ces deux automates soient identiques.

Deuxièmement, construisons conformément à la construction de la proposition 5.1 un automate non hiérarchique $\flat A' = (\flat Q', \Gamma', \flat\delta', i', f')$ de même langage que A'. En ajoutant à $\flat A'$ les états de contrôle de B et en y réintégrant l'ensemble des transitions contraintes de A, on obtient un automate alternant de piles $\flat A = (\flat Q, \Gamma', \flat\delta, (i' \wedge i_B), f')$, où $\flat Q = \flat Q' \cup Q_B$. Par construction, les états de contrôles de $\flat Q'$ are sont de la forme $q_n \dots q_k$ où $k \in [1,n]$ et chaque q_i est un état de contrôle d'un automate de niveau k apparaissant dans A'. On définit $\flat\delta$ comme l'union de δ_B et de l'ensemble de toutes les règles $s \xrightarrow{x} t$ telles que :

1. $\bar{p}pr \xrightarrow{x} \bar{p}q \in \flat\delta'$, $s = \bar{p}pr$, $t = (\bar{p}q \wedge u)$, $X =]$ et $p \longleftrightarrow CD(q,u)$ où C apparaît dans A et r est un état de contrôle de D,
2. $\bar{p}p \xrightarrow{x} \bar{p}q' \in \flat\delta'$, $s = \bar{p}p$, $t = (\bar{p}q \wedge u)$, $X = a$, et $p \longleftrightarrow Ca(q,u)$ où C est un automate de niveau 1 apparaissant dans A,
3. $s \xrightarrow{x} t \in \flat\delta'$ dans tous les autres cas.

On définit à présent le langage accepté par A comme le langage accepté par l'automate alternant $\flat A$ que nous venons de définir, selon la notion habituelle d'acceptation par automate alternant : $L(A) = L(\flat A)$ (rappelons que l'état initial de $\flat A$ est $i' \wedge i_B$).

L'idée intuitive est assez simple. Supposons que A est un automate hiérarchique B-contraint de niveau n, et B reconnaît lui aussi des piles de niveau n. Premièrement, on requiert que toutes les piles acceptées par A le soient aussi par B : $L(A) \subseteq L(B)$. Ensuite, pour tout calcul de A où une transition de la forme $p \xrightarrow{D} (q,r)$ intervient, la partie restant de l'entrée doit être acceptée à la fois par A en reprenant le calcul dans l'état q et par B depuis l'état initial r. Bien sûr, lorsqu'on remplace D par un mot de son langage, ceci peut déclencher de nouvelles vérifications par B.

Il faut remarquer malgré tout que les automates hiérarchiques contraints, dont nous avons fourni une transformation en automates alternants ordinaires, ne sont en fait pas plus expressifs que des automates ordinaires (mais exponentiellement plus concis).

Proposition 5.11. *Les automates hiérarchiques contraints acceptent des langages réguliers.*

5.4.2 Analyse d'accessibilité contrainte

Théoreme 5.12. *Étant donné un n-HCFP H et des ensembles réguliers S et C de piles de niveau n, l'ensemble $pre^*_H[C](S)$ est régulier et calculable de manière effective.*

Pour résoudre ce problème, nous proposons une version modifiée de la construction du paragraphe précédent utilisant des automates contraints. Soit $d = (a,o)$ une règle de transition d'HCFP, $A = (Q_A, \Gamma, \delta, i, f)$ et $A' = (Q_A, \Gamma, \delta', i, f)$

deux automates hiérarchiques de piles de niveau k contraints par un automate non hiérarchique $B = (Q_B,\Gamma',\delta_B,i_B,f_B)$ de piles de niveau n acceptant C (avec $n \geq k$). On définit une transformation $T^B_{d_j}(A)$, très similaire à T_{d_j}, mis à part le fait que nous avons besoin d'ajouter des transitions alternantes permettant de s'assurer qu'aucune des nouvelles piles introduites par la transformation n'est acceptée par A' si elle n'est pas effectivement la transformation d'une pile précédemment acceptée par B (cf. figure 5.2). Si $\mathrm{l}(d) < k$, on propage la transformation au premier automate de niveau $k-1$ sur chaque chemin :

$$\delta' = \{\, i \xrightarrow{T^B_d(C)} (p,q) \mid i \longleftrightarrow AC(p,q) \,\} \cup \{\, p \xrightarrow{C} (p',q') \in \delta \mid p \neq i \,\}.$$

Si $\mathrm{l}(d) = n$, on distingue trois cas selon la nature de d:

1. Si $d = (a,push^w_1)$, alors

$$\delta' = \delta \cup \Big\{\, i \xrightarrow{a} (p,q) \mid i \longleftrightarrow A_i w(p,q') \quad \wedge \quad \exists q_1,q \in Q_B, \\ \mathrm{l}(q_1) = \mathrm{l}(q) = 0,\ i_B \longleftrightarrow B[^n q_1 \longleftrightarrow Bwq \,\Big\}.$$

2. Si $d = (a,push_k)$, alors pour $m = n-k+1$ et $C = (C_1 \times C_2) \times (B_1 \times B_2) \times A^{(k-1)}_a$,

$$\delta' = \delta \cup \Big\{\, i \xrightarrow{C} (p,q) \mid i \longleftrightarrow A_iC_1 \longleftrightarrow A_iC_2(p,q') \quad \wedge \quad \exists\, q_1,q_2,q \in Q_B, \\ \mathrm{l}(q_1) = \mathrm{l}(q_2) = \mathrm{l}(q) = k-1,\ i_B \longleftrightarrow B[^m q_1 \overset{B_1}{\underset{B}{\rightsquigarrow}} q_2 \overset{B_2}{\underset{B}{\rightsquigarrow}} q \,\Big\}.$$

3. Si $d = (a,pop_k)$, alors pour $m = n-k+1$,

$$\delta' = \delta \cup \Big\{\, i \xrightarrow{A^{(k-1)}_a} (i,q) \mid \exists q \in Q_B,\ \mathrm{l}(q) = k-1,\ i_B \longleftrightarrow B_1[^m q \,\Big\}.$$

Supposons que $H = (\Gamma,\delta)$ avec $\delta = \{\, d_0,\ldots,d_{l-1} \,\}$. Soit un automate A tel que $S = L(A)$, nous considérons la séquence $(A_i)_{i\geq 0}$ définie par $A_0 = A^B$ (l'automate B-contraint de même états de contrôle et transitions que A, dont le langage est $L(A) \cap L(B)$) et pour tous $i \geq 0$ et $j = i \mod l$, $A_{i+1} = T^B_{d_j}(A_i)$. Par définition de T^B_d, le nombre d'états dans chaque A_i ne varie pas, et comme le nombre d'états dans B est fini les arguments de terminaison du lemme 5.5 sont toujours valides. Il est ensuite relativement direct d'étendre les démonstrations des lemmes 5.8 et 5.9 au cas contraint.

Lemme 5.13 (Terminaison)**.** *Pour tout automate hiérarchique A et automate non hiérarchique B de piles de niveau n, et pour tout n-HCFP $H = (\Gamma,\delta)$, la séquence (A^B_i) définie par rapport à A et B atteint $T^B_H(A)$ en un nombre fini d'étapes :*

$$\exists k \geq 0,\ \forall d \in \delta,\ T^B_d(A^B_k) = A^B_k.$$

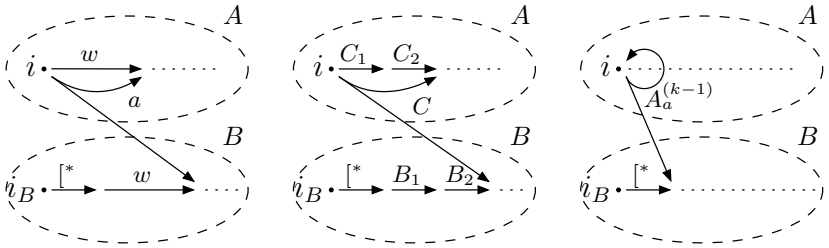


FIG. 5.2 – *La transformation* $T_d^B(A)$ *pour* $d = (a,push_1^w)$, $(a,push_k)$ *et* (a,pop_k).

Démonstration. L'algorithme qui permet de calculer est similaire à l'algorithme précédent pour $T_H(A)$, mis à part qu'il étiquette certaines des transitions de A_i^B par un état de contrôle de B. Comme le nombre de tels états reste fixe pendant l'intégralité du calcul, ceci n'affecte en rien le caractère borné du calcul ainsi que la finitude du nombre de transition total pouvant être ajoutées. □

Lemme 5.14 (Consistance)**.** $\forall i$, $L(A_i^B) \subseteq pre_H^*[C](S)$.

Démonstration. Par définition de $L(A_i^B)$, $L(A_i^B) \subseteq L(A_i)$ pour tout i. Donc, d'après le lemme 5.8, on a déjà $L(A_i^B) \subseteq pre_H^*(S)$. Raisonnons par récurrence sur i. Par définition des automates contraints, $L(A_0^B) = L(A) \cap C$, donc $L(A_0^B) \subseteq pre_H^*[C](S)$. Supposons la propriété vraie au rang i, et considérons l'automate A_{i+1}^B. Partout où la transformation T_d^B ajoute une transition dans A_i^B pour obtenir A_{i+1}^B, les transitions alternantes induites dans $\flat A_{i+1}^B$ assurent que chaque pile étiquetant un chemin acceptant dans l'automate est bien issu de la transformation d'une pile acceptée par B. De cette façon, on s'assure qu'aucun élément de $pre_H^*(S) \setminus pre_H^*[C](S)$ n'est ajouté à A_{i+1}^B.

Par exemple, supposons que $d = (a,push_1^w)$ et qu'une certaine pile s est acceptée par $\flat A_{i+1}^B$ grâce à une a-transition nouvellement créée par T_d^B. Selon la définition de T_d^B, cette transition est de la forme $p \xrightarrow{a} (q,r)$, où r est un état de contrôle de B accessible par un chemin étiqueté par $\lceil^n w$. Donc, si on pose $w(s) = \lceil^n a\, w'$, pour que s soit accepté par $\flat A_{i+1}^B$, il faut aussi que $s' = \lceil^n w\, w'$ soit accepté par B depuis l'état r. Le même type de raisonnement s'applique également aux autres opérations. □

Lemme 5.15 (Complétude)**.** $\forall i$, $S_i^C \subseteq L(A_i^B)$.

Démonstration. Par définition, $L(A_0^B) = S \cap C = S_0^C$. Supposons la propriété vraie au rang i, et considérons une pile $s \in S_{i+1}^C \setminus S_i^C$. Soit d l'opération telle que $S_{i+1}^C = S_i^C \cup (d(S_i^C) \cap C)$. Par définition il existe une pile $s' \in S_i^C$ telle que $s' = d(s)$, et par hypothèse de récurrence s' est acceptée par A_i^B. De plus, comme à la fois s et s' sont dans C, elles sont toutes deux acceptées par B. Comme on

l'a vu au lemme 5.9, la transformation T_d ajoute une nouvelle transition $p_0 \xrightarrow{C} q$ créant en particulier un chemin étiqueté par s. Les contraintes supplémentaires que T_d^B ajoute à cette transition, et à tous les chemins dans A_{i+1}^B in général, interdit à tout chemin étiqueté par un certain r utilisant cette transition d'être acceptant à moins qu'à la fois r et $d(r)$ ne soient acceptés par B. C'est la cas pour s et s', et donc $s \in L(A_{i+1}^B)$. □

Cette construction plus générale nous permet d'étendre le théorème 5.10 au fragment plus général $\mathsf{E}(\mathsf{U},\mathsf{X})$ de CTL, où les formules sont à présent construites grâce à l'opérateur modal EU à la place de EF.

Théoreme 5.16. *Étant donné un HCFP H et une formule close φ de $\mathsf{E}(\mathsf{U},\mathsf{X})$, l'ensemble des configurations de H satisfaisant φ est régulier et calculable de façon effective.*

Conclusion

Cette thèse présente plusieurs contributions au domaine des graphes infinis de présentation finie. Nous donnons un bref résumé des résultats obtenus, ainsi que des questions et perspectives ouvertes par les travaux de chaque chapitre de ce document.

Résumé des résultats et questions ouvertes

Systèmes de réécriture de termes à dérivation rationnelle

Dans le chapitre 3, nous avons étudié trois familles de systèmes de réécriture de termes dont la relation de dérivation est rationnelle, et peut en particulier être décrite au moyen d'un formalisme fini. Ces familles sont formées des systèmes descendants, ascendants et suffixes. Les résultats présentés fournissent une caractérisation interne immédiate de plusieurs familles de graphes infinis, ainsi que des applications possibles dans l'analyse de systèmes paramétrés. Plusieurs questions ouvertes émergent de ce travail :

1. Les dérivations de nos familles de systèmes sont définies comme des sous-familles des relations rationnelles de termes de Raoult [Rao97]. Est-il possible de caractériser directement ces familles de relations à l'aide d'automates ou de transducteurs appropriés ?
2. Quelles sont les propriétés de fermeture de chacune de ces familles de relations ? En particulier, les dérivations des systèmes suffixes forment-elles une algèbre de Boole ?
3. Comment les familles de graphes de réécriture des systèmes que nous avons étudiés se comparent-elles aux familles de graphes existantes ? Pouvons-nous donner d'autres caractérisations internes ou externes de ces familles ? Quelles sont les propriétés structurelles de ces graphes (par exemple, leurs traces) ?
4. Il est montré dans [Cau00] que les dérivations des systèmes gauches ou droits coïncident avec les relations rationnelles sur les mots. Peut-on de la même façon caractériser les relations rationnelles de Raoult à l'aide de systèmes de réécriture ?

Accepteurs infinis pour les langages contextuels

Le chapitre 4 donne une étude approfondie de plusieurs familles d'accepteurs infinis pour les langages contextuels déterministes et non-déterministes. En particulier, le paragraphe 4.2 présente de nouvelles preuves auto-suffisantes du fait que les traces des graphes rationnels sont les langages contextuels, même en se restreignant aux graphes de degré sortant fini à sommet initial unique. Nous avons également caractérisé les traces des graphes synchronisés de degré fini depuis un sommet initial unique comme les langages des machines linéairement bornées à nombre linéaire de retours, et avons prouvé que dans le cas de degré borné leurs langages forment une sous-famille des langages contextuels déterministes. Enfin, nous avons proposé une condition sur les ensembles de transducteurs telle que les traces des graphes rationnels correspondants décrivent précisément la famille des langages contextuels déterministes.

Dans le paragraphe 4.3, nous avons défini la famille des graphes linéairement bornés, dont nous avons donné trois caractérisations internes différentes. Ils sont les graphes de type Cayley des systèmes de réécriture décroissants, les graphes de transductions contextuelles incrémentales et les graphes de transition des machines linéairement bornées. Une conséquence directe de cette dernière présentation est que les langages de ces graphes sont les langages contextuels. D'autres propriétés incluent la fermeture de cette famille par restriction aux sommets accessibles ou restriction à un ensemble contextuel de sommets. Nous avons enfin étudié le cas des langages contextuels déterministes et démontré qu'ils sont les traces des graphes de transition déterministes des machines linéairement bornées dont tout calcul termine.

Ce chapitre est clos par une comparaison des deux familles de graphes considérées, dont la conclusion est que les graphes rationnels de degré borné sont inclus dans les graphes linéairement bornés de degré borné mais que la réciproque est fausse. Puisque les graphes linéairement bornés de degré borné acceptent tous les langages contextuels, ceci nous permet de définir une hiérarchie de familles de graphes infinis de degré borné dont les traces (depuis un sommet unique) forment la hiérarchie de Chomsky.

Les résultats présentés dans ce chapitre donnent lieu à plusieurs questions et extensions possible.

1. A la fois les graphes linéairement bornés et les graphes rationnels ont une théorie au premier ordre indécidable. Ce n'est pas le cas des graphes synchronisés, mais il n'est pas clair que leurs traces incluent tous les langages contextuels depuis un sommet initial unique et avec un degré fini. Est-il possible de définir une famille de graphes de théorie au premier ordre décidable acceptant tous les langages contextuels ?
2. Une façon possible de répondre à la question précédente serait de démontrer

que les graphes synchronisés de degré fini acceptent bel et bien tous les langages contextuels depuis un sommet initial unique. Ce problème se réduit au problème de l'égalité des langages contextuels avec les langages acceptés par les machines linéairement bornées en un nombre linéaire de retour, qui est une question ouverte non triviale de théorie de la complexité.

3. Il semble faisable d'étendre la plupart des constructions du paragraphe 4.2 aux familles correspondantes de graphes sur les termes (graphes terme-automatiques et terme-rationnels, en utilisant notamment les transductions rationnelles de Raoult [Rao97]).
4. La comparaison entre les graphes linéairement bornés et d'autres familles de graphes, comme les familles de graphes sur les termes mentionnées à l'instant ou la hiérarchie C. devrait être étudiée. Une question connexe est de fournir d'autres présentations (en particulier externes) de ces graphes.

Analyse d'accessibilité de processus hors-contexte d'ordre supérieur

Enfin, nous avons proposé au chapitre 5 une réflexion préliminaire sur l'extension de techniques de vérification symbolique existantes à la classe des automates à pile d'ordre supérieur. Nous avons montré que l'analyse symbolique d'accessibilité de [BEM97] peut être étendue à une sous-famille de ces systèmes, que nous avons appelés processus hors-contexte d'ordre supérieur. Ces processus correspondent aux automates à pile d'ordre supérieur ne possédant qu'un unique état de contrôle. Notre technique utilise les langages réguliers de mots pour représenter de façon symbolique des ensembles de configurations, et calcule (avec une complexité non élémentaire) l'ensemble des prédécesseurs d'un tel ensemble par un processus hors-contexte d'ordre supérieur donné.

On peut citer plusieurs autres options pour poursuivre cette étude. Indépendamment de notre travail, Carayol a défini dans [Car05] une notion d'ensembles rationnels de piles d'ordre supérieur. Cette notion fournit directement une technique d'analyse symbolique d'accessibilité avant et arrière naturelle pour les automates à pile d'ordre supérieur. Cependant, il montre également que la plupart des opérations nécessaires (comme le test du vide d'un ensemble rationnel) est non-élémentaire. Il serait sensé de rechercher à la lumière de ces nouveaux résultats si une amélioration significative de la complexité peut être obtenue en simplifiant le modèle étudié, par exemple à nos processus hors-contexte. Plus généralement, il serait intéressant d'examiner la structure des graphes de transition de ces processus et de la comparer avec le cas général. Enfin, il reste le problème de déterminer si ces processus peuvent trouver un usage comme outils de modélisation pour une classe non triviale de programmes, au même titre que les automates à pile d'ordre supérieur peuvent être utilisés pour modéliser des programmes fonctionnels d'ordre supérieur [Dam82, KNU02].

Remarques finales

Notre point de vue est que l'étude structurelle des graphes infinis constitue une approche générale et élégante à la compréhension de nombreuses variétés différentes de systèmes. Voir un système de transition comme un graphe peut être considéré comme aussi naturel et utile que d'utiliser des graphes finis pour représenter le comportement de machines à espaces d'états finis. Les résultats concernant la caractérisation alternative d'une même famille de graphes par plusieurs formalismes peuvent en outre entraîner des conséquences intéressantes pour chacune de leurs familles de représentants, ainsi que des démonstrations simplifiées pour certaines de leurs propriétés.

De plus, les graphes infinis possèdent de nombreux liens avec d'autres domaines de l'informatique théorique ou appliquée. Ils sont un outil de modélisation très puissant, et il est à espérer que ce domaine de recherche en développement aidera à fournir de nouveaux outils pour la mise au point de méthodes de vérification de systèmes infinis. De plus, les graphes infinis vus comme accepteurs de langages formels ont de nombreux liens avec la théorie des langages, comme l'a soutenu par exemple Urvoy [Urv03] en établissant un rapport précis entre familles de graphes infinis et familles de langages.

Annexe A

Accepteurs finis pour les langages contextuels

Le but de cette annexe est de présenter les différentes familles d'accepteurs finis des langages contextuels ainsi que les transformations permettant de passer de l'un à l'autre, dans l'espoir de convaincre le lecteur que ces transformations très simples et purement syntaxiques. Il vise aussi à établir des résultats de complexité fins pour les traductions entre chaque formalisme.

A.1 LBM non étiquetées

Les définitions habituelles des machines linéairement bornées possèdent des transitions non étiquetées, démarrent leurs calculs dans une configuration où le mot d'entrée figure sur la bande de travail de la machine, et ne sont capables ni d'ajouter ni de retirer des cellules de la bande. Malgré ces différences, les deux définitions des LBM coïncident. Dans la proposition suivante, nous nous rapportons à notre définition des LBM comme à des LBM étiquetées.

Proposition A.1. *Un langage est contextuel (déterministe) si et seulement si il est accepté par une machine linéairement bornée étiquetée (déterministe).*

Démonstration. Considérons tout d'abord un langage contextuel L accepté par une machine linéairement bornée non étiquetée et terminante N (cf. proposition 1.10). Nous décrivons les grandes lignes de la construction d'une LBM étiquetée $M = (\Gamma,\Sigma,[,],Q,q_0,F,\delta)$ acceptant L.

Soient q_A,q_R et q_S trois états de contrôle dans Q. Dans une configuration de la forme $[wq_X]$ avec $w \in \Sigma^*$ et $q_X \in \{q_A,q_R\}$, M lit une lettre d'entrée $a \in \Sigma$ et avance jusqu'à la configuration $[waq_s]$. Puis elle simule N sur wa en n'utilisant que des ε-transitions, tout en conservant une sauvegarde de wa par codage sur l'alphabet. Si N accepte (resp. rejette) wa, M restaure wa sur sa bande et entre dans

la configuration $[waq_A]$ (resp. $[waq_R]$). En prenant $F = \{q_A\}$ et $q_0 = q_A$ si ε est dans L et $q_0 = q_R$ sinon, il est facile de voir que l'ensemble des mots acceptés par M depuis $[q_0]$ est précisément L. Notons qu'un comportement non-déterministe ne peut apparaître dans M qu'en simulant N. Donc si N est déterministe, alors M l'est aussi.

Réciproquement, soit M une LBM étiquetée acceptant un langage $L \subset \Sigma^*$. Nous décrivons une machine non étiquetée N acceptant L en utilisant deux bandes : une bande d'entrée et une bande de travail. Il est bien connu que ce modèle est équivalent aux LBM non étiquetées à une seule bande (voir par exemple [HU79]). À chaque bande correspond un ensemble d'états de contrôle : l'ensemble des états de travail Q_Γ contient l'ensemble Q des états de N et l'ensemble d'états de lecture Q_Σ est réduit à $\{q_i,q_A\}$ qui sont respectivement état initial et état acceptant.

La machine N lisant le mot $w \in \Sigma^*$ démarre dans la configuration $([q_iw],[q_0])$. De toute configuration de la forme $([w_1q_iw_2],[uqv])$ avec $w_1,w_2 \in \Sigma^*$, $q \in Q$ et $u,v \in \Gamma^*$, N peut simuler de façon non-déterministe toute ε-transition de M applicable dans la configuration actuelle de la bande de travail. Les règles de suppression sont simulées en décalant tout le contenu à droite de la tête de lecture d'une case vers la gauche. De plus, M peut simuler de façon non-déterministe une transition étiquetée par a avec $a \in \Sigma$ pourvu que la tête de la bande de lecture fasse face à une cellule contenant le symbole a, auquel cas elle est décalée d'un cran sur la droite. Les règles d'insertion sont simulées en décalant tout le contenu de la bande à droite de la tête de lecture d'un cran sur la droite.

La machine M passe dans l'état acceptant q_A si la tête d'entrée fait face au symbole de bordure] et l'état de travail est un état acceptant de M. Il découle de la construction de N que w est accepté si et seulement si il est aussi accepté par M. De plus, si M est déterministe alors N l'est aussi. □

A.2 Automates cellulaires unidirectionnels

Dans un automate cellulaire, la prochaine valeur d'une cellule dépend de sa valeur courante ainsi que de la valeur de ses voisins gauche et droit. Si nous n'autorisons la valeur suivante d'un cellule à dépendre que de sa valeur courante et de celle de son voisin de gauche, on parle d'automate cellulaire unidirectionnel, qui ont la même expressivité (cf. théorème 4.6).

Définition A.2. Un *automate cellulaire unidirectionnel* C est un sextuplet $(\Gamma,\Sigma,F,[,],\delta)$ avec

- Γ et $\Sigma \subseteq \Gamma$ les alphabets de travail et d'entrée,
- $F \subseteq \Gamma$ l'alphabet d'acceptation,
- [et] des symboles de délimitation n'appartenant pas à Γ,

– $\delta \subseteq (\{[\} \cup \Gamma) \times \Gamma \times \Gamma) \cup (\Gamma \times \{]\} \times \{]\})$ la fonction de transition.

Pour tous $u,v \in \Gamma^+$, une configuration $c' = [u]$ est un successeur d'une configuration $c = [v]$ si $|c| = |c'| = n$ et si pour tout $i \in [1,n-1]$, $(c(i),c(i+1),c'(i+1)) \in \delta$. Dans la suite, on représentera la transition $(A,B,C) \in \delta$ par :

A	B
	C

A.3 Résultats d'expressivité généraux

Nous présentons quelques résultats bien connus de simulation entre les divers accepteurs mentionnés jusqu'ici : les machines de Turing linéairement bornées, les automates cellulaires, les automates cellulaires unidirectionnels et les systèmes de pavage.

Théoreme A.3. *Pour tout langage L, les propositions suivantes sont équivalentes :*

1. *L est un langage contextuel,*
2. *L est reconnu par une machine de Turing linéairement bornée,*
3. *L est reconnu par un automate cellulaire,*
4. *L est reconnu par un automate cellulaire unidirectionnel,*
5. *L est reconnu par un système de pavage.*

Démonstration. Nous démontrons les implications suivantes :

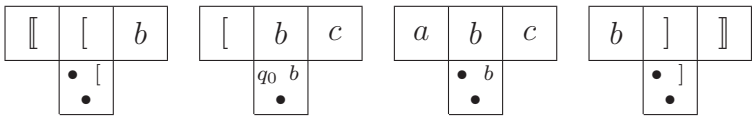
$1 \Leftrightarrow 2$: Comme dit précédemment, les langages contextuels furent originellement caractérisés comme les langages engendrés par des grammaires croissantes, ou grammaires contextuelles. Le lien avec les machines de Turing linéairement bornées a été établi dans [Kur64].

$2 \Rightarrow 3$: Étant donnée une machine linéairement bornée $M = (\Gamma,\Sigma,[,],Q,q_0,F,\delta)$, on peut supposer sans perte de généralité que M est telle que depuis toute configuration il existe une transition vers une configuration bloquante et non acceptante. Ce comportement peut être obtenu en ajoutant un nouvel état $q_\perp$ à Q et toutes les transitions possibles depuis les états de F vers $q_\perp$. Nous définissons un automate cellulaire $C = (\Gamma',\Sigma,[\![,]\!],F',\delta')$ acceptant le langage $[L(M)]$. Soit

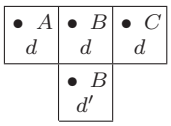
$$\Gamma' = \Big(\Sigma \ \cup \ \{[,]\}\Big) \ \cup \ \Big((Q \cup \bullet) \times \big(\Gamma \cup \{[,]\}\big) \times (\delta \cup \bullet)\Big)$$

l'alphabet de travail de C. Chaque lettre dans Γ' est soit une lettre de Σ, un caractère de bordure [ou], ou un triplet (c,A,d) avec c un état de contrôle

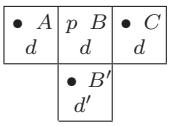
de M ou le symbole spécial $\bullet$, A l'un des symboles de bande de M et d la dernière transition utilisée ou le symbole $\bullet$.
Premièrement, on spécifie l'ensemble des transitions δ' de C. Nous avons besoin de transitions permettant d'encoder la configuration initiale de M partant du mot d'entrée :



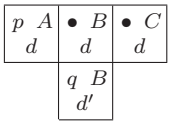
où $b \in \Sigma$ et a,$c \in \Sigma \cup \{[,]\}$. Nous avons maintenant besoin de décrire comment l'automate cellulaire simule une étape de calcul de la machine :



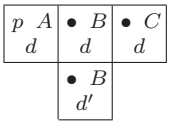
avec d' une transition dans δ,



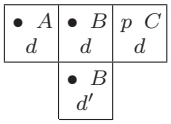
avec $d' = pB \to qB'\pm \in \delta$,



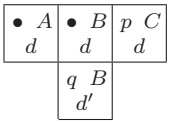
avec $d' = pA \to qA'+ \in \delta$,



avec $d' = pA \to qA'- \in \delta$,

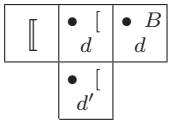


avec $d' = pC \to qC'+ \in \delta$,

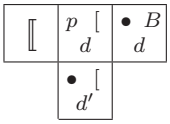


avec $d' = pC \to qC'- \in \delta$,

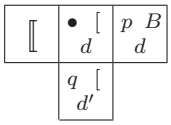
où d peut être une transition quelconque ou $\bullet$, $B \in \Gamma$ et A,$C \in \Gamma \cup \{[,]\}$. Nous devons aussi traiter les bordures de la configuration grâce aux règles suivantes (seul le cas gauche est présenté, l'autre cas étant symétrique) :



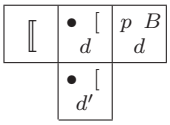
avec d' une transition dans δ,



avec $d' = p[\to q[+ \in \delta$,



avec $d' = pB \to qB'- \in \delta$,



avec $d' = pB \to qB'+ \in \delta$,

où d peut être une transition quelconque et $B \in \Gamma$. Il est impossible d'arrêter le calcul dès que la configuration c contient un états de F, car cette configuration n'a pas encore été vérifiée et peut se révéler mal formée (c'est à dire

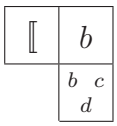
que c pourrait ne pas faire partie de l'ensemble $\big((Q \cup \bullet) \times (\Gamma \cup \{[,]\}) \times \{d\}\big)^*$ pour un certain $d \in \delta$), mais nous pouvons accepter si l'état précédent appartenait à F. Donc l'ensemble F' d'états finaux de C est l'ensemble des symboles de Γ' dont la transition est de la forme $q_f A \to pB\epsilon$ avec $q_f \in F$ (qui existe nécessairement grâce à l'ajout de l'état $q_\perp$).

Soit π le morphisme de $(\Gamma')^*$ dans $(Q \cup \Gamma)^*$ défini par $\pi(\bullet,A,d) = A$ et $\pi(q,A,d) = qA$, il est facile de vérifier qu'il existe un calcul $c_0,c_1,\ldots,c_n,c_{n+1}$ de C si et seulement si $\pi(c_1),\ldots,\pi(c_n)$ est un calcul de M. De plus, si le calcul $c_0,c_1 \ldots c_n,c_{n+1}$ est acceptant (c'est à dire si $c_{n+1} \in [(F')^+]$), alors $\pi(c_n)$ contient un état final et $\pi(c_1),\ldots,\pi(c_n)$ est aussi acceptant. La réciproque est vraie grâce à l'ajout de l'état $q_\perp$.

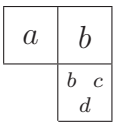
Par conséquent le langage reconnu par C est $[L(M)]$. Un automate cellulaire C' reconnaissant précisément $L(M)$ peut facilement être construit à partir de C en étendant l'alphabet de travail. Sa complexité en temps est égale à celle de M plus deux.

$3 \Rightarrow 4$: Soit $C = (\Gamma,\Sigma,F,[,],\delta)$ un automate cellulaire. Construisons un automate cellulaire unidirectionnel $C' = (\Gamma',\Sigma,F',[\![,]\!],\delta')$ équivalent, avec $\Gamma' = \Sigma \cup ((\Gamma \cup \{\,]\,\}) \times \Gamma \times (\Gamma \cup \{\,\square\,\}))$. Un symbole de Γ' est soit un symbole d'entrée de C soit un triplet (p,q,r) où p représente la valeur courante d'une cellule, q correspond à une devinette de la valeur courante du voisin droit de la cellule, et r est sa valeur suivante possible ou le symbole $\square \notin \Gamma$.

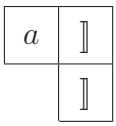
L'ensemble de transitions δ' est décrit ci-après. Premièrement, nous devons deviner la valeur courante du voisin droit de chaque cellule et calculer une valeur suivante possible en fonction de cette valeur devinée, de la valeur du voisin gauche, de la valeur actuelle et de l'ensemble de règles δ. On peut aussi indiquer qu'il n'y a pas de valeur suivante à l'aide du symbole $\square$:



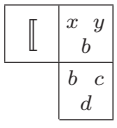
avec $([,b,c,d) \in \delta$ ou $d = \square$



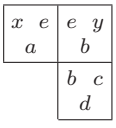
avec $(a,b,c,d) \in \delta$ ou $d = \square$



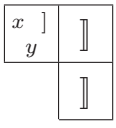
Ensuite, nous continuons à appliquer ce principe en vérifiant de surcroît que chaque valeur devinée à l'étape précédente était bien correcte :



avec $([,b,c,d) \in \delta$ ou $d = \square$



avec $(a,b,c,d) \in \delta$ ou $d = \square$



où $x \in \Gamma$ et $y \in \Gamma \cup \{\,]\,\}$. Le calcul réussit quand un symbole acceptant de C apparaît comme valeur courante de toutes les cellules (soit un symbole $(f,x,y) \in \Gamma'$ avec $f \in F$). En d'autres termes, $F' = F \times (\Gamma \cup \{\,]\,\}) \times (\Gamma \cup \{\square\})$.
Soit un calcul acceptant $\rho = c_1, \ldots, c_n$ de C avec $c_1 = [w]$. Pour tout $i \in [1,n]$, définissons $c'_i \in \Gamma'^*$ comme suit. Pour tout $i \in [1,n]$, soient $c'_i(1) = [\![$ et $c'_i(|c_1|) =]\!]$. Pour tous $i < n$ et $j \in [2,|c_1|-1]$, soit $c'_i(j) = (c_i(j), c_i(j+1), c_{i+1}(j+1))$. Enfin pour tout $j \in [2,|c_1|-1]$, soit $c'_n(j) = (c_{n-1}(j), c_{n-1}(j+1), \square)$. Par construction de δ', la séquence $[\![w]\!], c'_1 \ldots, c'_n$ est un calcul acceptant de C'. Donc $L(C) \subseteq L(C')$.
Réciproquement, soit π la projection suivante : $\pi([\![) = [$, $\pi(]\!]) =]$, et pour tout $\gamma = (p,q,r) \in \Gamma' \setminus \Gamma$, $\pi(\gamma) = p$. Alors si $c_1, c_2, \ldots c_n$ est un calcul acceptant de C' avec $c_1 = [\![w]\!]$, alors $\pi(c_2), \ldots, \pi(c_n)$ est un calcul acceptant de C avec $\pi(c_2) = [w]$. Donc $L(C') \subseteq L(C)$.
Il s'ensuit que C' est équivalent à C. De plus, si C accepte en temps $f(n)$ alors C' accepte en temps $f(n)+1$.

$4 \Rightarrow 5$: Soit $C = (\Gamma,\Sigma,F,[,],\delta)$ un automate cellulaire unidirectionnel. Considérons le système de pavage $S = (\Gamma,\Sigma,\#,\Delta)$ tel que Δ contienne les tuile

$\#$ B / $\#$ C	$\#$ $\#$ / $\#$ B	pour tout	$[$ B / $\;$ C	$\in \delta$	
	A B / X C	pour tout	A B / $\;$ C	$\in \delta$	
A $\#$ / X $\#$	$\#$ $\#$ / A $\#$	pour tout	A $]$ / $\;$ $]$	$\in \delta$	
	$\#$ $\#$ / A B	pour tout	$A,B \in \Sigma$		

où A,B,C et X sont des lettres de Γ. Ces tuiles assurent que les images acceptées forment des calculs valides de C. Il reste à s'assurer que la dernière est de la forme $[F^+]$. Pour cela, nous ajoutons les tuiles

$\#$ f / $\#$ $\#$	f f' / $\#$ $\#$	f' $\#$ / $\#$ $\#$

où f et f' sont dans F.

$5 \Rightarrow 2$: Soit $S = (\Gamma,\Sigma,\#,\Delta)$ un système de pavage. Nous construisons une machine linéairement bornée $M = (\Gamma,\Sigma,[,],Q,p_0,F,\delta)$ reconnaissant $L(S)$. Première-

ment, soit

$$Q = \{p_B^A, q_B^A \mid A,B \in \Gamma \cup \#\} \cup \{p_{0A}, p_{fA}, q_{fA} \mid A \in \Gamma \cup \#\} \cup \{p_0, p_f, q_f\}$$

avec $F = \{p_f, q_f\}$. Nous définissons à présent δ. Tout d'abord, il nous faut un ensemble de règles qui nous permettent de vérifier que le mot d'entrée est une première ligne possible d'une image de $P(S)$. Pour tous $C,D \in \Sigma$, on a :

$$p_0 D \to p_{0D} D+ \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline \# & D \\ \hline \end{array} \in \Delta,$$

$$p_{0C} D \to p_{0D} D+ \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline C & D \\ \hline \end{array} \in \Delta,$$

$$p_{0C}\,] \to q_{\#}^{\#}\,]- \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline C & \# \\ \hline \end{array} \in \Delta.$$

Ces règles permettent de s'assurer qu'il existe un calcul partiel monotone de M de la forme $[p_0 w] \xrightarrow{*} [w_1 q_{\#}^{\#} w_2]$ où $w_1 w_2 = w$ et $w_2 \in \Sigma$ si et seulement si w est une première ligne possible d'une image de $P(S)$.
Ensuite, nous devons transformer en un passage de la tête de lecture, soit de la droite vers la gauche soit l'inverse, une configuration représentant une ligne d'une image de $P(S)$ en une autre configuration représentant une possible ligne suivante dans la même image, en changeant de direction quand la tête rencontre une bordure. Pour tous $A,B,C,D \in \Gamma$, on a donc :

$$\left.\begin{array}{r} p_{\#}^{\#} B \to p_D^B D+ \\ q_D^B\,[\to p_{\#}^{\#}\,[+ \end{array}\right\} \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline \# & B \\ \hline \# & D \\ \hline \end{array} \in \Delta,$$

$$\left.\begin{array}{r} p_C^A B \to p_D^B D+ \\ q_D^B A \to q_C^A C- \end{array}\right\} \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline A & B \\ \hline C & D \\ \hline \end{array} \in \Delta,$$

$$\left.\begin{array}{r} p_C^A\,] \to q_{\#}^{\#}\,]- \\ q_{\#}^{\#} A \to q_C^A C- \end{array}\right\} \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline A & \# \\ \hline C & \# \\ \hline \end{array} \in \Delta.$$

Ces règles assurent qu'il existe un calcul partiel monotone $[p_{\#}^{\#} u] \longrightarrow [v_1 q_{\#}^{\#} v_2]$ de M si et seulement si u et $v_1 v_2$ sont une paire de lignes adjacentes dans au moins une image de $P(S)$. Réciproquement, c'est aussi le cas de $u_1 u_2$ et v pour tout calcul partiel monotone $[u_1 q_{\#}^{\#} u_2] \longrightarrow [p_{\#}^{\#} v]$. Cela signifie qu'entre deux changements de direction de la tête, M remplace le contenu courant

de la bande vue comme une ligne d'image par une ligne adjacente possible selon les tuiles de Δ.
Enfin, nous devons être en mesure de reconnaître une dernière ligne possible d'image de $P(S)$, et passer dans un état acceptant de la machine si c'est le cas. Une fois encore, ceci peut être effectué dans les deux sens, selon le nombre de lignes reconnues jusque là :

$$\left.\begin{array}{r} p^{\#}_{\#}B \to p_{f_B}B+ \\ q_{f_B}[\to p_f[+ \end{array}\right\} \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline \# & B \\ \hline \# & \# \\ \hline \end{array} \in \Delta,$$

$$\left.\begin{array}{r} p_{f_A}B \to p_{f_B}B+ \\ q_{f_B}A \to q_{f_A}A- \end{array}\right\} \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline A & B \\ \hline \# & \# \\ \hline \end{array} \in \Delta,$$

$$\left.\begin{array}{r} p_{f_A}] \to q_f]- \\ q^{\#}_{\#}A \to q_{f_A}A- \end{array}\right\} \quad \text{pour tout} \quad \begin{array}{|c|c|}\hline A & \# \\ \hline \# & \# \\ \hline \end{array} \in \Delta.$$

Ces dernières règles assurent qu'il existe un calcul partiel monotone dans M depuis $[p^{\#}_{\#}u]$ jusqu'à $[u_1q_fu_2]$ ou depuis $[u_1q^{\#}_{\#}u_2]$ jusqu'à $[p_fu]$ si et seulement si $u = u_1u_2$ est une dernière ligne possible d'une image de $P(S)$. En rassemblant toutes ces observations, nous concluons que $L(M) = L(S)$. De plus, si un mot w est la frontière d'une image de taille m dans $P(S)$, alors il est accepté par M en m retours.

□

A.4 Meilleures bornes de complexité

Les démonstrations précédentes sont syntaxiques, et visent à exprimer l'importante similarité entre ces quatre familles d'accepteurs. Cependant, pour obtenir des résultats de simulation plus fins, on a besoin de mettre en place des constructions légèrement plus complexes quand on traduit une machine de Turing en automate cellulaire.

Proposition A.4. *Les simulations suivantes relient les machines de Turing linéairement bornées, les automates cellulaires, les automates cellulaires unidirectionnels et les systèmes de pavage.*

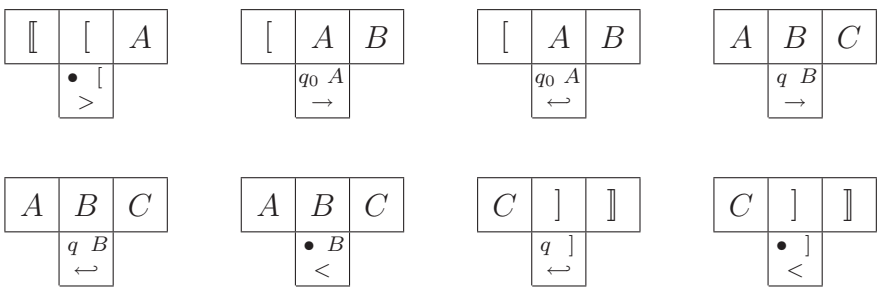
1. *Une machine de Turing linéairement bornée M travaillant en $f(n)$ retours peut être simulée par un automate cellulaire C travaillant en temps $f(n)+2$.*
2. *Un automate cellulaire C travaillant en temps $f(n)$ peut être simulé par un automate cellulaire unidirectionnel C' travaillant en temps $f(n)+1$.*
3. *Un automate cellulaire unidirectionnel C travaillant en temps $f(n)$ peut être simulé par un système de pavage S travaillant en hauteur $f(n)$.*

4. Un système de pavage travaillant en hauteur $f(n)$ peut être simulé par une machine de Turing linéairement bornée M travaillant en $f(n)$ retours.

Démonstration. 1. Soit $M = (\Gamma,\Sigma,[,],Q,q_0,F,\delta)$ une machine de Turing linéairement bornée travaillant en $f(n)$ retours. On définit un automate cellulaire $C = (\Gamma',\Sigma,F',[\![,]\!],\delta')$ reconnaissant le langage $[L(M)]$ en temps $f(n)+2$. L'idée de la construction est, pour tout calcul possible, d'encoder en une seule étape de l'automate cellulaire toutes les étapes de calcul de M entre deux renversements de la tête successifs. Donc, la longueur d'un calcul de C simulant un calcul ρ de M devrait être égal au nombre de retours dans ρ, plus deux pour l'étape d'initialisation et l'étape finale.

Pour s'assurer de la correction, il nous faut vérifier que des cellules adjacentes s'accordent sur la direction dans laquelle la tête se déplace, que l'état de contrôle atteint par M dans chaque cellule (ou leur absence) obéit aux règles de transition de la machine, et qu'aucun état de contrôle n'apparaît dans les cellules qui ne sont pas situées entre le dernier et le prochain point de renversement de la tête de la machine. De plus, nous devons vérifier que les symboles apparaissant dans chaque cellule sont en accord avec les symboles et l'état de contrôle apparaissant dans la même cellule à l'étape précédente. Pour cela, il suffit de définir l'alphabet Γ' utilisé par C comme suit. Soit $\Gamma' = \Sigma \cup (Q \cup \{\bullet\}) \times \Gamma \times \{\hookleftarrow, \hookrightarrow, \leftarrow, \rightarrow, <, >\}$. Nous utilisons les symboles $\hookleftarrow$ et $\hookrightarrow$ pour noter un renversement de la tête (retour), $\leftarrow$ et $\rightarrow$ pour un déplacement monotone de la tête, et $<$ et $>$ pour marquer les cellules qui sont à droite (resp. à gauche) de la région active de la bande. Chaque cellule contient également un état de contrôle de M, ou $\bullet$ quand la cellule n'est pas atteinte par la tête à ce moment.

À présent, nous pouvons décrire l'ensemble de règles δ'. Nous ne donnons pas l'ensemble complet de règles, mais illustrons plutôt la construction sur quelques cas représentatifs. Tout d'abord, nous devinons le comportement de la machine de Turing entre le début du calcul et le premier retour. Les règles permettant ceci incluent



où A,B et C sont des symboles de Γ et q un état de contrôle dans Q. Maintenant que la première séquence monotone de mouvements été devinée,

on doit vérifier sa validité et décrire le reste du calcul. Comme il existe un nombre considérable de règles à énumérer, nous nous contentons de décrire certaines des conditions que nos règles doivent observer. Une règle (W,X,Y,Z) est dans δ' si et seulement si elle satisfait *toutes* les contraintes suivantes (et leurs versions duales) :

(a) Si W (resp. X) est de la forme $(\bullet,A,>)$ alors X (resp. Y) doit être de la forme $(\bullet,B,>)$, $(p,B,\rightarrow)$ ou $(p,B,\hookleftarrow)$.

(b) Si W (resp. X) est de la forme $(p,A,\rightarrow)$ alors X (resp. Y) doit être de la forme $(q,B,\rightarrow)$ ou $(q,B,\hookleftarrow)$ avec $pA \rightarrow qA'+ \in \delta$.

(c) Si W (resp. X) est de la forme $(p,A,\hookleftarrow)$ alors X (resp. Y) doit être de la forme $(\bullet,B,<)$.

(d) Si W (resp. X) est de la forme $(\bullet,A,<)$ alors X (resp. Y) doit être de la forme $(\bullet,B,<)$.

(e) Si X est de la forme $(p,A,\rightarrow)$ alors Z doit être de la forme $(\bullet,A',>)$ ou $(p',A',\leftarrow)$ ou $(p',A',\hookrightarrow)$ avec $pA \rightarrow qA'+ \in \delta$.

(f) Si Y est de la forme $(p,A,\hookleftarrow)$ alors Z doit être de la forme $(q,B,\rightarrow)$ ou $(q,B,\hookrightarrow)$ avec $pA \rightarrow qB- \in \delta$.

Comme dit précédemment, les règles doivent aussi satisfaire toutes les contraintes duales (c'est à dire des contraintes portant sur X par rapport à Y, W par rapport à X et Z par rapport à W quand les flèches sont dans l'autre sens). Il existe aussi quelques contraintes simples concernant les délimiteurs $[\![$ et $]\!]$. Enfin, nous devons donner un dernier ensemble de règles afin de pouvoir vérifier que la configuration précédente est correcte et qu'elle contient un état final. Ceci n'est pas détaillé ici mais ne pose pas de difficulté.

2. La construction donnée dans la démonstration du théorème A.3 traduit un automate cellulaire travaillant en temps $f(n)$ en un automate cellulaire unidirectionnel travaillant en temps $f(n)+1$.
3. La construction donnée dans la démonstration du théorème A.3 traduit un automate cellulaire unidirectionnel travaillant en temps $f(n)$ en un système de pavage de hauteur d'images $f(n)+1$.
4. La construction donnée dans la démonstration du théorème A.3 traduit un système de pavage de hauteur d'images $f(n)$ en une machine linéairement bornée travaillant en $f(n)$ retours.

□

A.5 Cas déterministe

Enfin, nous présentons quelques résultats moins classiques analogues aux simulations précédentes dans le cas des langages contextuels déterministes, qui sont

à la base caractérisés comme les langages acceptés par les machines linéairement bornées déterministes. Des notions correspondantes de déterminisme peuvent être définies pour tous les autres types d'accepteurs, au sens où cela leur permet d'accepter exactement cette classe de langages.

Théoreme A.5. *Pour tout langage L, les propositions suivantes sont équivalentes :*

1. *L est un langage contextuel déterministe,*
2. *L est accepté par une machine linéairement bornée déterministe,*
3. *L est accepté par un automate cellulaire déterministe,*
4. *L est accepté par un système de pavage déterministe.*

Démonstration. Nous démontrons les implications suivantes :

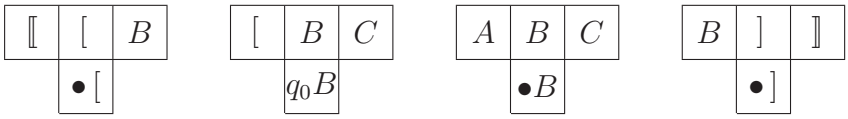
$1 \Leftrightarrow 2$: Par définition des langages contextuels déterministes.

$2 \Rightarrow 3$: Soit $M = (\Gamma,\Sigma,[,],Q,q_0,F,\delta)$ une machine linéairement bornée déterministe. On peut supposer sans perte de généralité que M est telle que toute configuration acceptante est bloquante. On définit un automate cellulaire déterministe $C = (\Gamma',\Sigma,\{\perp\},[\![,]\!],\delta')$ reconnaissant le langage $[L(M)]$. Soit

$$\Gamma' = \Big(\Sigma \ \cup \ \{[,]\}\Big) \ \cup \ \Big((Q \cup \bullet) \times \big(\Gamma \cup \{[,]\}\big)\Big)$$

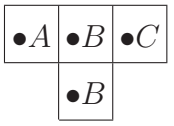
l'alphabet de travail de C. Chaque lettre de Γ' est soit une lettre dans Σ, un symbole de bordure, ou un couple (c,A) où c est un état de contrôle de M ou le symbole spécial $\bullet$, et A est un symbole de bande de M. Les transitions δ' de C sont définies comme dans le cas standard.

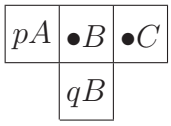
Premièrement, nous avons besoin de transitions pour encoder la configuration initiale de M à partir du mot d'entrée :



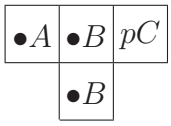
où $B \in \Sigma$ et $A,C \in \Sigma \cup \{[,]\}$.

Nous devons aussi décrire comment l'automate cellulaire simule une étape de calcul de la machine :

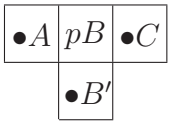




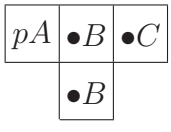
avec $pA \to qA'+ \in \delta$,



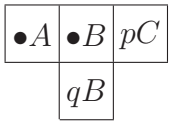
avec $pC \to qC'+ \in \delta$,



avec $pB \to qB'\pm \in \delta$,

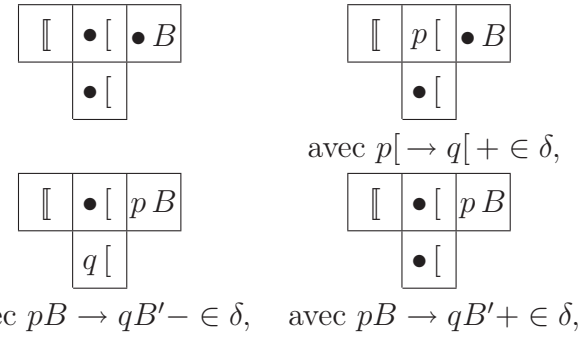


avec $pA \to qA'- \in \delta$,



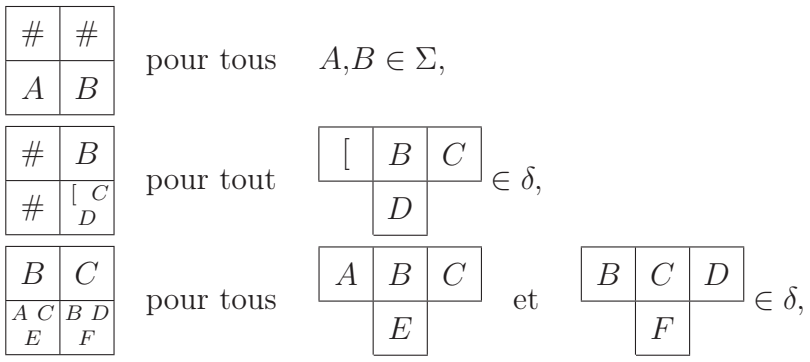
avec $pC \to qC'- \in \delta$,

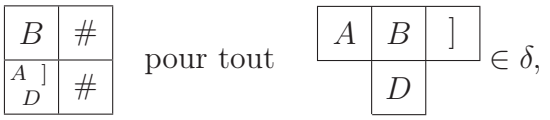
où $B \in \Gamma$ et A,$C \in \Gamma \cup \{[,]\}$.
Il nous faut également prendre en compte les bords de la configuration avec les règles suivantes (seul un cas est présenté, l'autre étant symétrique) :



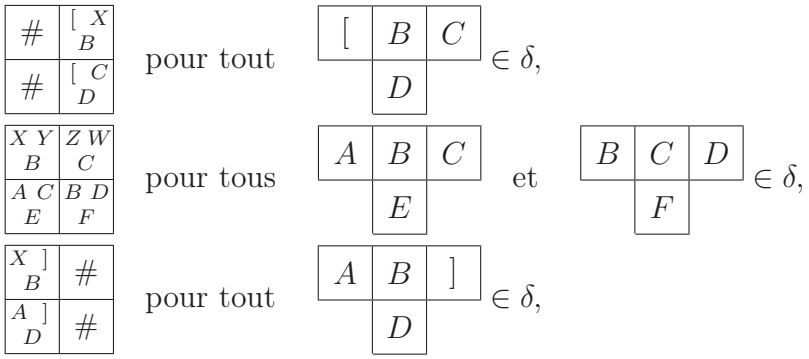
où $B \in \Gamma$. Contrairement au cas général, il est maintenant possible de stopper le calcul dès que la configuration courante contient un état dans F, en faisant apparaître de façon déterministe et en propageant un symbole spécial $\perp$ jusqu'à atteindre une configuration de la forme $[\![\perp^*]\!]$. Ceci ne sera pas détaillé ici. Notons que C est effectivement déterministe.
Soit π le morphisme de $(\Gamma')^*$ à $(Q \cup \Gamma)^*$ défini par $\pi(\bullet,A) = A$ et $\pi(q,A) = qA$, il est possible de démontrer qu'il existe un calcul $c_0,c_1 \dots c_n$ de C avec c_n contenant un symbole de F si et seulement si le calcul $\pi(c_1), \dots \pi(c_n)$ est acceptant pour M. Il s'ensuit que le langage reconnu par C est $[L(M)]$. Un automate déterministe C' reconnaissant exactement $L(M)$ peut ensuite facilement être construit à partir de C.

$3 \Rightarrow 4$: Soit $C = (\Gamma,\Sigma,F,[,],\delta)$ un automate cellulaire déterministe. Nous construisons un système de pavage déterministe équivalent $S = (\Gamma',\Sigma,\#,\Delta)$. L'ensemble de symboles Γ' est égal à $\Sigma \cup \Gamma^3$. Un symbole de Γ' est soit une lettre de Σ soit un triplet (l,n,r) de symboles de Γ où l (resp. r) représente la valeur du voisin gauche (resp. droit) de la cellule courante à l'étape précédente et c représente la valeur courante de la cellule. L'ensemble de tuiles Δ contient les tuiles initiales

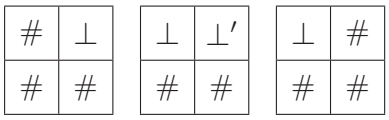




ainsi que les tuiles



et enfin un ensemble de tuile détectant une configuration finale dans $[F^+]$:



où $\perp$ et $\perp' \in \Gamma \times F \times \Gamma$.

Il est facile de vérifier que le système de pavage S est déterministe. Soit p une image dans $P(S)$ et soient $r_0, \ldots, r_m$ les lignes de $p_\#$, nous montrons que la frontière de p est acceptée par C. Si l'on définit la projection π par $\pi(\#w\#) = [w]$ pour $w \in \Sigma^+$ et $\pi(\#(x_1,y_1,z_1)\ldots(x_n,y_n,z_n)\#) = [y_1 \ldots y_n]$, par construction $\pi(r_1), \ldots, \pi(r_{m-1})$ est un calcul acceptant de C.

Réciproquement, si w est accepté par C alors il existe un calcul acceptant $c_1, \ldots, c_n$. Soit p l'image de taille $(n, |c_1| - 2)$ sur Γ' définie par $p(1,i) = c_1(i+1)$ pour $i \in [2, |c_1| - 2]$ et par $p(j,i) = (c_{j-1}(i-2), c_j(i-1), c_{j-1}(i))$ pour $i \in [2, |c_1| - 2]$ et $j \in [2, n]$. Par construction on a $p_\# \in P(S)$ et donc $w \in L(S)$.

$4 \Rightarrow 2$: Soit $S = (\Gamma, \Sigma, \#, \Delta)$ un système de pavage déterministe, on construit une machine linéairement bornée déterministe M reconnaissant $L(S)$. La construction donnée pour le cas général ne s'étend pas directement au cas déterministe, puisqu'un calcul peut échouer en essayant d'engendrer une ligne suivante possible à partir de la ligne courante. Dans ce cas, on doit être en mesure de s'assurer que le calcul n'échoue pas tant que l'on n'est pas sûr que le mot est rejeté.

Une ligne r dans $\#\Gamma^+\# \cup \#^+$ est successeur d'une ligne r' si $|r'| = |r|$ et les tuiles de l'image p formée des lignes r' et r sont dans Δ.

En partant d'un mot w, la machine M vérifie que $\#w\#$ est successeur de $\#^{|w|+2}$ (si ce n'est pas le cas, M rejette). La ligne courante de M est maintenant $\#w\#$ et M devra calculer de façon déterministe le successeur de cette ligne. Elle énumère toutes les lignes suivantes possibles en commençant par $\#^{|w|+2}$, vérifiant à chaque fois la ligne convient. Si $\#^{|w|+2}$ est sélectionnée, M accepte le mot w. Si la ligne courante n'admet pas de successeur alors M rejette w. Sinon, comme S est déterministe il existe exactement un successeur r pour la ligne courante, que M inscrit alors sur sa bande avant de réitérer le processus de recherche de successeur.
Par construction, M accepte un mot $w \in \Sigma^+$ si et seulement si $w \in L(S)$.
□

Bibliographie

[ABJ98] P. Abdulla, A. Bouajjani et B. Jonsson : On-the-fly analysis of systems with unbounded, lossy fifo channels. *In Proceedings of the 10th International Conference on Computer Aided Verification (CAV 1998)*, volume 1427 de *Lecture Notes in Computer Science*, pages 305–318. Springer, 1998.

[AEM04] R. Alur, K. Etessami et P. Madhusudan : A temporal logic of nested calls and returns. *In Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 de *Lecture Notes in Computer Science*, pages 467–481. Springer, 2004.

[AJMD02] P. Abdulla, B. Jonsson, P. Mahata et J. D'Orso : Regular tree model checking. *In Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, volume 2404 de *Lecture Notes in Computer Science*, pages 555–568. Springer, 2002.

[AN82] A. Arnold et M. Nivat : Comportements de processus. *In Colloque de l'Association Francaise pour la Cybernétique Économique et Théorique: Les Mathématiques de l'Informatique (AFCET 1982)*, pages 35–68, 1982.

[Bar97] K. Barthelmann : On equational simple graphs. Rapport technique 9, Universität Mainz, Institut für Informatik, 1997.

[Bar98] K. Barthelmann : When can an equational simple graph be generated by hyperedge replacement? *In Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS 1998)*, volume 1450 de *Lecture Notes in Computer Science*, pages 543–552. Springer, 1998.

[BC04] Y. Bertot et P. Castéran : *Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An Eatcs Series. Springer, 2004.

[BCS96] O. Burkart, D. Caucal et B. Steffen : Bisimulation collapse and the process taxonomy. *In Proceedings of the 7th International Conference on Concurrency Theory (CONCUR 1996)*, volume

1119 de *Lecture Notes in Computer Science*, pages 247–262. Springer, 1996.

[BEM97] A. Bouajjani, J. Esparza et O. Maler : Reachability analysis of pushdown automata: Application to model-checking. *In Proceedings of the 8th International Conference on Concurrency Theory (CONCUR 1997)*, volume 1243 de *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[Ber79] J. Berstel : *Transductions and Context-Free Languages*. Leitfäden der angewandten Mathematik und Mechanik. Teubner, 1979.

[BG00] A. Blumensath et E. Grädel : Automatic structures. *In Proceedings of the 15th IEEE Symposium on Logic in Computer Science (LICS 2000)*, pages 51–62. IEEE, 2000.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems et P. Wolper : The power of qdds. *In Proceedings of the 4th International Symposium on Static Analysis (SAS 1997)*, volume 1302 de *Lecture Notes in Computer Science*, pages 172–186. Springer, 1997.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson et T. Touili : Regular model checking. *In Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, volume 1855 de *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

[BK89] J. Bergstra et J. Klop : Process theory based on bisimulation semantics. *In Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (REX Workshop)*, volume 354 de *Lecture Notes in Computer Science*, pages 50–122. Springer, 1989.

[Blu01] A. Blumensath : Prefix-recognizable graphs and monadic second order logic. Rapport technique AIB-06-2001, RWTH Aachen, 2001.

[BM04] A. Bouajjani et A. Meyer : Symbolic reachability analysis of higher-order context-free processes. *In Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2004)*, volume 3328 de *Lecture Notes in Computer Science*, pages 135–147. Springer, 2004.

[Bou01] A. Bouajjani : Languages, rewriting systems, and verification of infinite-state systems. *In Proceedings of the 28th International Colloquium on Automata, Languages, and Programming (ICALP 2001)*, volume 2076 de *Lecture Notes in Computer Science*, pages 24–39. Springer, 2001.

[BT02] A. Bouajjani et T. Touili : Extrapolating tree transformations. *In Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, volume 2404 de *Lecture Notes in Computer Science*, pages 539–554. Springer, 2002.

[Büc62] J. Büchi : On a decision method in restricted second order arithmetic. *In Proceedings of the 1960 International Congress of Logic, Methodology and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

[Cac02] T. Cachat : Symbolic strategy synthesis for games on pushdown graphs. *In Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP 2002)*, volume 2380 de *Lecture Notes in Computer Science*, pages 704–715. Springer, 2002.

[Car01] A. Carayol : Notions of determinism for rational graphs. Mémoire de maîtrise, ENS Lyon, France, 2001.

[Car05] A. Carayol : Regular sets of higher-order pushdown stacks. *In Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS 2005)*, Lecture Notes in Computer Science. Springer, 2005. To appear.

[Cau92] D. Caucal : On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[Cau95] D. Caucal : Bisimulation of context-free grammars and of pushdown automata. *In* A. Ponse, M. de Rijke et Y. Venema, éditeurs : *Modal Logic and Process Algebra*, volume 53 de *CSLI Lecture Notes*, pages 85–106. Stanford, 1995.

[Cau96] D. Caucal : On infinite transition graphs having a decidable monadic theory. *In Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP 1996)*, volume 1099 de *Lecture Notes in Computer Science*, pages 194–205. Springer, 1996.

[Cau00] D. Caucal : On word rewriting systems having a rational derivation. *In Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2000)*, volume 1784 de *Lecture Notes in Computer Science*, pages 48–62. Springer, 2000.

[Cau02] D. Caucal : On infinite terms having a decidable monadic theory. *In Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS 2002)*, volume 2420 de *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002.

[Cau03] D. Caucal : On the transition graphs of Turing machines. *Theoretical Computer Science*, 296:195–223, 2003.

[CC77] P. Cousot et R. Cousot : Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. *In Conference Record of the Fourth ACM*

Symposium on Principles of Programming Languages (POPL 1977), pages 238–252, 1977.

[CC03] A. Carayol et T. Colcombet : On equivalent representations of infinite structures. *In Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP 2003)*, volume 2719 de *Lecture Notes in Computer Science*, pages 599–610. Springer, 2003.

[CDG+] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison et M. Tommasi : Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata.

[CES86] E. Clarke, A. Emerson et P. Sistla : Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[Cho59] N. Chomsky : On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959.

[Cho82] L. Chottin : Langages algébriques et systèmes de réécriture rationnelle. *Informatique Théorique et Applications*, 16(2):93–112, 1982.

[Chu36] A. Church : A note on the entscheidungsproblem. *The Journal of Symbolic Logic*, 1(1):40–41, 1936.

[CK98] H. Calbrix et T. Knapik : A string-rewriting characterization of Muller and Schupp's context-free graphs. *In Proceedings of the 18th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1998)*, volume 1530 de *Lecture Notes in Computer Science*, pages 331–342. Springer, 1998.

[CK01] D. Caucal et T. Knapik : An internal presentation of regular graphs by prefix-recognizable graphs. *Theoretical Computer Science*, 34:299–336, 2001.

[CK02a] D. Caucal et T. Knapik : A Chomsky-like hierarchy of infinite graphs. *In Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS 2002)*, volume 2420 de *Lecture Notes in Computer Science*, pages 177–187. Springer, 2002.

[CK02b] B. Courcelle et T. Knapik : The evaluation of first-order substitution is monadic second-order compatible. *Theoretical Computer Science*, 281(1-2):177–206, 2002.

[CM05a] A. Carayol et A. Meyer : Context-sensitive languages, rational graphs and determinism. To appear (preliminary version available at `http://www.liafa.jussieu.fr/~ameyer/`), 2005.

[CM05b] A. Carayol et A. Meyer : Linearly bounded infinite graphs. *In Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS 2005)*, volume 3618 de *Lecture Notes in Computer Science*, pages 180–191. Springer, 2005.

[CM05c] A. Carayol et A. Meyer : Linearly bounded infinite graphs (long version). Submitted (preliminary version available at `http://www.liafa.jussieu.fr/~ameyer/`), 2005.

[Col02] T. Colcombet : On families of graphs having a decidable first order theory with reachability. *In Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP 2002)*, volume 2380 de *Lecture Notes in Computer Science*, pages 98–109. Springer, 2002.

[Col04] T. Colcombet : *Représentations et propriétés de structures infinies*. Thèse de doctorat, Université de Rennes 1, France, 2004.

[Cou89] B. Courcelle : The monadic second-order logic of graphs, II: Infinite graphs of bounded width. *Mathematical System Theory*, 21:187–221, 1989.

[Cou90] B. Courcelle : Graph rewriting: An algebraic and logic approach. *In* J. van Leeuwen, éditeur : *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 193–242. Elsevier, 1990.

[Cou94] B. Courcelle : Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

[Cou97] B. Courcelle : The expression of graph properties and graph transformations in monadic second-order logic. *In* G. Rozenberg, éditeur : *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapitre 5, pages 313–400. World Scientific, 1997.

[CW03] A. Carayol et S. Wöhrle : The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. *In Proceedings of the 23rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2003)*, volume 2914 de *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

[Dam82] W. Damm : The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

[DHLT90] M. Dauchet, T. Heuillard, P. Lescanne et S. Tison : Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems. *Information and Computation*, 88(2):187–201, 1990.

[DJ90] N. Dershowitz et J.-P. Jouannaud : Rewrite systems. *In* J. van Leeuwen, éditeur : *Handbook of Theoretical Computer Science, Vol. B*, pages 243–320. Elsevier, 1990.

[DT85] M. Dauchet et S. Tison : Decidability of the confluence of finite ground term rewrite systems. *In Proceedings of the 5th International Symposium on Fundamentals of Computation Theory, (FCT 1985)*, volume 199 de *Lecture Notes in Computer Science*, pages 80–89. Springer, 1985.

[DTHL87] M. Dauchet, S. Tison, T. Heuillard et P. Lescanne : Decidability of the confluence of ground term rewriting systems. *In Proceedings of the 2nd IEEE Symposium on Logic in Computer Science (LICS 1987)*, pages 353–359. IEEE, 1987.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith et S. Schwoon : Efficient algorithm for model checking pushdown systems. *In Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, volume 1885 de *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000.

[EM65] C. Elgot et J. Mezei : On relations defined by finite automata. *IBM Journal of Research and Development*, 9:47–68, 1965.

[Eme90] E. A. Emerson : Temporal and modal logic. *In Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. Elsevier, 1990.

[Eng83] J. Engelfriet : Iterated pushdown automata and complexity classes. *In Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC 1983)*, pages 365–373. ACM, ACM Press, 1983.

[EW05] Z. Esik et P. Weil : Algebraic recognizability of regular tree languages. *Theoretical Computer Science*, 2005. To appear.

[For00] L. Fortnow : Diagonalization. *Bulletin of the European Association for Theoretical Computer Science*, 71:102–112, 2000.

[FS93] C. Frougny et J. Sakarovitch : Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1): 45–82, 1993.

[GR96] D. Giammarresi et A. Restivo : *Handbook of Formal Languages*, volume 3, chapitre Two-dimensional languages. Springer, 1996.

[GS97] F. Gécseg et M. Steinby : *Handbook of Formal Languages*, volume 3, chapitre 1, pages 1–68. Springer, 1997.

[GV98] P. Gyenizse et S. Vágvölgyi : Linear generalized semi-monadic rewrite systems effectively preserve recognizability. *TCS*, 194(1–2): 87–122, 1998.

[HJJ+95] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe et A. Sandholm : Mona: Monadic second-order logic in practice. *In Proceedings of the 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1995)*, volume 1019 de *Lecture Notes in Computer Science*, pages 89–110. Springer, 1995.

[HU79] J. Hopcroft et J. Ullman : *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Imm88] N. Immerman : Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[JW96] D. Janin et I. Walukiewicz : On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. *In Proceedings of the 7th International Conference on Concurrency Theory (CONCUR 1996)*, volume 1119 de *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

[Kle56] S. Kleene : Representation of events in nerve nets and finte automata. *In Automata studies*, volume 34 de *Annals of Mathematics Studies*, pages 3–40. Princeton University Press, 1956.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli et E. Shahar : Symbolic model checking with rich assertional languages. *In Proceedings of the 9th International Conference on Computer Aided Verification (CAV 1997)*, volume 1254 de *Lecture Notes in Computer Science*, pages 424–435. Springer, 1997.

[Kni64] J. D. Mc Knight : Kleene quotient theorem. *Pacific Journal of Mathematics*, pages 1343–1352, 1964.

[KNU02] T. Knapik, D. Niwinski et P. Urzyczyn : Higher-order pushdown trees are easy. *In Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2002)*, volume 2303 de *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

[Kob69] K. Kobayashi : Classification of formal languages by functional binary transductions. *Information and Control*, 15(1):95–109, 1969.

[KP99] T. Knapik et É. Payet : Synchronized product of linear bounded machines. *In Proceedings of the 12th International Symposium on Fundamentals of Computation Theory, (FCT 1999)*, volume 1684 de *Lecture Notes in Computer Science*, pages 362–373. Springer, 1999.

[Kur64] S. Kuroda : Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, 1964.

[Löd02] Christof Löding : Ground tree rewriting graphs of bounded tree width. *In Proceedings of the 19th Annual Symposium on Theoretical*

Aspects of Computer Science (STACS 2002), volume 2285 de *Lecture Notes in Computer Science*, pages 559–570. Springer, 2002.

[LS97] M. Latteux et D. Simplot : Context-sensitive string languages and recognizable picture languages. *Information and Computation*, 138(2):160–169, 1997.

[LST98] M. Latteux, D. Simplot et A. Terlutte : Iterated length preserving rational transductions. *In Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS 1998)*, volume 1450 de *Lecture Notes in Computer Science*, pages 286–295. Springer, 1998.

[LW01] K. Lodaya et P. Weil : Rationality in algebras with a series operation. *Information and Computation*, 171(2):269–293, 2001.

[MD98] J. Mazoyer et M. Delorme : Cellular automata: a parallel model. *In Cellular Automata as Language Recognizers*, pages 153–180. Kluwer Academic Publishings, 1998.

[Mey02] A. Meyer : Sur des sous-familles de graphes rationnels. Mémoire de D.E.A., Université de Rennes 1, France, 2002.

[Mey04] A. Meyer : On term rewriting systems having a rational derivation. *In Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2004)*, volume 2987 de *Lecture Notes in Computer Science*, pages 378–392. Springer, 2004.

[Mil89] R. Milner : *Communication and Concurrency*. Prentice Hall International, 1989.

[Mor00] C. Morvan : On rational graphs. *In Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2000)*, volume 1784 de *Lecture Notes in Computer Science*, pages 252–266. Springer, 2000.

[Mor01] C. Morvan : *Les graphes rationnels*. Thèse de doctorat, Université de Rennes 1, France, 2001.

[MR05] Christophe Morvan et Chloé Rispal : Families of automata characterizing context-sensitive languages. *Acta Informatica*, 41(4-5):293–314, 2005.

[MS85] D. E. Muller et P. E. Schupp : The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[MS01] C. Morvan et C. Stirling : Rational graphs trace context-sensitive languages. *In Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001)*, vo-

lume 2136 de *Lecture Notes in Computer Science*, pages 548–559. Springer, 2001.

[Mye79] G. Myers : *The Art of Software Testing*. John Wiley & Sons, 1979.

[Nec97] G. Necula : Proof-carrying code. *In Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1997)*, pages 106–119. ACM Press, 1997.

[NPW02] T. Nipkow, L. Paulson et M. Wenzel : *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[ORS92] S. Owre, J. Rushby, et N. Shankar : PVS: A prototype verification system. *In Proceedings of the 11th International Conference on Automated Deduction (CADE 1992)*, volume 607 de *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer, 1992.

[Pay00] É. Payet : *Thue Specifications, Infinite Graphs and Synchronized Product*. Thèse de doctorat, Université de la Réunion, 2000.

[Pen74] M. Penttonen : One-sided and two-sided context in formal grammars. *Information and Control*, 25(4):371–392, 1974.

[Pnu77] A. Pnueli : The temporal logic of programs. *In Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS 1977)*, pages 46–57. IEEE Computer Society Press, 1977.

[Rab69] M. Rabin : Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[Rao91] J.-C. Raoult : A survey of tree transductions. Rapport technique 1410, Inria, avril 1991.

[Rao97] J.-C. Raoult : Rational tree relations. *Bulletin of the Belgian Mathematics Society*, 4:149–176, 1997.

[Ris02] C. Rispal : The synchronized graphs trace the context-sensitive languages. *In Proceedings of the 4th International Workshop on Verification of Infinite-State Systems (INFINITY 2002)*, volume 68 de *Electronic Notes in Theoretical Computer Science*, 2002.

[Sak03] J. Sakarovitch : *Éléments de théorie des automates*. Éditions Vuibert, 2003.

[Sti98] C. Stirling : The joys of bisimulation. *In Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS 1998)*, volume 1450 de *Lecture Notes in Computer Science*, pages 142–151. Springer, 1998.

[Sti00] C. Stirling : Decidability of bisimulation equivalence for pushdown processes. Rapport technique EDI-INF-RR-0005, School of Informatics, University of Edinburgh, 2000.

[TKS00] T. Takai, Y. Kaji et H. Seki : Right-linear finite path overlapping term rewriting systems effectively preserve recognizability. *In Proceedings of the 11th International Conference on Rewriting Techniques and Applications (RTA 2000)*, volume 1833 de *Lecture Notes in Computer Science*, pages 246–260. Springer, 2000.

[Tra50] B. Trakhtenbrot : The impossibilty of an algorithm for the decision problem for finite models. *Doklady Akademii Nauk SSR*, 70:569–572, 1950.

[Tur36] A. Turing : On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[Urv03] T. Urvoy : *Familles abstraites de graphes*. Thèse de doctorat, Université de Rennes 1, France, June 2003.

[vM04] D. van Melkebeek : Time-space lower bounds for NP-complete problems. *Current Trends in Theoretical Computer Science*, pages 265–291, 2004.

[VW86] M. Vardi et P. Wolper : An automata-theoretic approach to automatic program verification (preliminary report). *In Proceedings of the 1st Symposium on Logic in Computer Science (LICS 1986)*, pages 332–344. IEEE Computer Society, 1986.

[Wal96] I. Walukiewicz : Pushdown processes: Games and model checking. *In Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, volume 1102 de *Lecture Notes in Computer Science*, pages 62–74. Springer, 1996.

[Wal02] I. Walukiewicz : Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1–2):311–346, 2002.

[WB98] P. Wolper et B. Boigelot : Verifying systems with infinite but regular state spaces. *In Proceedings of the 10th International Conference on Computer Aided Verification (CAV 1998)*, volume 1427 de *Lecture Notes in Computer Science*, pages 88–97. Springer, 1998.

[Web96] A. Weber : Decomposing a k-valued transducer into k unambiguous ones. *Informatique Théorique et Applications*, 30(5):379–413, 1996.

[Wei04] P. Weil : Algebraic recognizability of languages. *In Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS 2004)*, volume 3153 de *Lecture Notes in Computer Science*, pages 149–175. Springer, 2004.

Résumé

Cette thèse s'inscrit dans l'étude de familles de graphes infinis de présentation finie, de leurs propriétés structurelles, ainsi que des comparaisons entre ces familles. Étant donné un alphabet fini Σ, un graphe infini étiqueté par Σ peut être caractérisé par un ensemble fini de relations binaires $(R_a)_{a\in\Sigma}$ sur un domaine dénombrable V quelconque. De multiples caractérisations finies de tels ensembles de relations existent, soit de façon explicite grâce à des systèmes de réécriture ou à divers formalismes de la théorie des automates, soit de façon implicite.

Après un survol des principaux résultats existants, nous nous intéressons plus particulièrement à trois problèmes. Dans un premier temps, nous définissons trois familles de systèmes de réécriture de termes dont nous démontrons que la relation de dérivation peut être représentée de façon finie. De ces résultats découlent plusieurs questions sur les familles de graphes infinis correspondantes. Dans un second temps, nous étudions deux familles de graphes dont les ensembles de traces forment la famille des langages contextuels, à savoir les graphes rationnels et les graphes linéairement bornés. Nous nous intéressons en particulier au cas des langages contextuels déterministes, ainsi qu'à la comparaison structurelle de ces deux familles. Enfin, d'un point de vue plus proche du domaine de la vérification, nous proposons un algorithme de calcul des prédécesseurs pour une famille d'automates à pile d'ordre supérieur.

Abstract

This thesis contributes to the study of families of finitely presented infinite graphs, their structural properties and their relations to each other. Given a finite alphabet Σ, a Σ-labeled infinite graph can be characterized as a finite set of binary relations $(R_a)_{a\in\Sigma}$ over an arbitrary countable domain V. There are many ways to finitely characterize such sets of relations, either explicitly using rewriting systems or formalisms from automata theory, either externally.

After giving an overview of the main results in this domain, we focus on three specific problems. In a first time, we define several families of term-rewriting systems whose derivation relation can be finitely represented. These results raise interesting questions concerning the corresponding families of infinite graphs. In a second time, we study two families of infinite graphs whose sets of traces (or languages) coincide with the well-known family of context-sensitive languages. They are the rational graphs and the linearly bounded graphs. We investigate the case of deterministic context-sensitive languages, and establish a structural comparison between these two families of graphs. Finally, in an approach closer to the concerns of the verification community, we propose a symbolic reachability algorithm for a class of higher-order pushdown automata.