Réécriture d'arbres de piles et traces de systèmes à compteurs

Vincent Penelle

Thèse de doctorat

pour l'obtention du grade de

Docteur de l'université Paris-Est

Spécialité Informatique

au titre de l'École Doctorale de Mathématiques
et des Sciences et Techniques de l'Information et de la Communication

Présentée et soutenue publiquement par

Vincent Penelle

le 20 novembre 2015

Réécriture suffixe d'arbres de piles et traces de systèmes à compteurs

Devant le jury composé par

Christof Löding	Rapporteur
Géraud Sénizergues	Rapporteur
Marie-Pierre Béal	Examinatrice
Didier Caucal	Examinateur
Thomas Colcombet	Examinateur
Jérôme Leroux	Examinateur
Antoine Meyer	Examinateur

Laboratoire d'informatique Gaspard-Monge
UMR 8049 LIGM
5, bd Descartes, Champs-sur-Marne, 77454 Marne-la-Vallée Cedex 2, France

Merci !

Réécriture suffixe d'arbres de piles et traces de systèmes à compteurs — Ground Stack Tree Rewriting Systems and Traces of Counter Systems

Résumé

Dans cette thèse, nous nous attachons à étudier des classes de graphes infinis et leurs propriétés, notamment celles de vérification de modèles, d'accessibilité et de langages reconnus.

Nous définissons une notion d'arbres de piles ainsi qu'une notion liée de réécriture suffixe, permettant d'étendre à la fois les automates à piles d'ordre supérieur et la réécriture suffixe d'arbres de manière minimale. Nous définissons également une notion de reconnaissabilité sur les ensembles d'opérations à l'aide d'automates qui induit une notion de reconnaissabilité sur les ensembles d'arbres de piles et une notion de normalisation des ensembles reconnaissables d'opérations analogues à celles existant sur les automates à pile d'ordre supérieur. Nous montrons que les graphes définis par ces systèmes de réécriture suffixe d'arbres de piles possèdent une FO-théorie décidable, en montrant que ces graphes peuvent être obtenu par une interprétation à ensembles finis depuis un graphe de la hiérarchie à piles.

Nous nous intéressons également au problème d'algébricité des langages de traces des systèmes à compteurs et au problème lié de la stratifiabilité d'un ensemble semi-linéaire. Nous montrons que dans le cas des polyèdres d'entiers, le problème de stratifiabilité est décidable et possède une complexité coNP-complète. Ce résultat nous permet ensuite de montrer que le problème d'algébricité des traces de systèmes à compteurs plats est décidable et coNP-complet. Nous donnons également une nouvelle preuve de la décidabilité des langages de traces des systèmes d'addition de vecteurs, préalablement étudié par Schwer.

Abstract

In this thesis, we study classes of infinite graphs and their properties, especially the model-checking problem, the accessibility problem and the recognised languages.

We define a notion of stack trees, and a related notion of ground rewriting, which is an extension of both higher-order pushdown automata and ground tree rewriting systems. We also define a notion of recognisability on the sets of ground rewriting operations through operation automata. This notion induces a notion of recognisability over sets of stack trees and a normalisation of recognisable sets of operations, similar to the known notions over higher-order pushdown automata. We show that the graphs defined by these ground stack tree rewriting systems have a decidable FO-theory, by exhibiting a finite set interpretation from a graph defined by a higher-order automaton to a graph defined by a ground stack tree rewriting system.

We also consider the context-freeness problem for counter systems, and the related problem of stratifiability of semi-linear sets. We prove the decidability of the stratifiability problem for integral polyhedra and that it is coNP-complete. We use this result to show that the context-freeness problem for flat counter systems is as well coNP-complete. Finally, we give a new proof of the decidability of the context-freeness problem for vector addition systems, previously studied by Schwer.

Table des matières

1 Introduction **1**
- 1.1 Historique et contributions . . . 4
- 1.2 Plan de la thèse . . . 8

2 Préliminaires **11**
- 2.1 Notions de base . . . 11
  - 2.1.1 Ensembles et sous-ensembles . . . 11
  - 2.1.2 Relations . . . 11
  - 2.1.3 Fonctions . . . 12
  - 2.1.4 Morphismes . . . 12
  - 2.1.5 Vecteurs de nombres . . . 13
  - 2.1.6 Matrices . . . 13
- 2.2 Mots et langages . . . 13
- 2.3 Graphes . . . 13
- 2.4 Automates finis et langages reconnaissables . . . 14
- 2.5 Logiques . . . 15
  - 2.5.1 Structures relationnelles . . . 15
  - 2.5.2 Logique du premier ordre . . . 15
  - 2.5.3 Logique du second ordre monadique . . . 17

3 Graphes infinis **19**
- 3.1 Présentations finies de graphes infinis . . . 19
  - 3.1.1 Isomorphismes de graphes . . . 19
  - 3.1.2 Questions classiques . . . 19
  - 3.1.3 Représentation interne . . . 21
  - 3.1.4 Transformations de graphes . . . 21
- 3.2 Graphes liés aux piles . . . 26
  - 3.2.1 Automates à pile . . . 26
  - 3.2.2 Automates à pile d'ordre supérieur . . . 30
  - 3.2.3 Un modèle équivalent : les systèmes de réécriture de piles . . . 36
- 3.3 Graphes liés aux arbres . . . 41
  - 3.3.1 Arbres . . . 41
  - 3.3.2 Systèmes de réécriture d'arbres . . . 42
  - 3.3.3 Automates d'arbres . . . 42
  - 3.3.4 Relations arbre-synchronisées . . . 45
  - 3.3.5 Graphes arbre-automatiques . . . 45
  - 3.3.6 Systèmes reconnaissables de réécriture suffixe d'arbres . . . 46
  - 3.3.7 Graphes de systèmes de réécriture d'arbres . . . 47

3.3.8 Hiérarchie arbre-automatiques 49
3.4 Récaputilatif 49

4 Systèmes de réécriture suffixe d'arbres de piles **51**
4.1 Définition du modèle 51
4.1.1 Arbres de piles 51
4.1.2 Opérations 52
4.2 Extensions des piles d'ordre supérieur et des arbres 56
4.3 Automates d'opérations 59
4.3.1 Définition 60
4.3.2 Propriétés de clôture 60
4.3.3 Réécriture reconnaissable et transducteurs suffixes 62
4.3.4 Automates normalisés 63
4.4 Graphes de réécriture d'arbres de piles 71
4.4.1 Notations et formules techniques 73
4.4.2 La formule δ 74
4.4.3 La formule ϕ_A associée à un automate normalisé A 75
4.4.4 La formule ψ_A associée à un automate normalisé A 79
4.5 Traces des graphes de réécritures d'arbres de piles 80

5 Traces de systèmes à compteurs **85**
5.1 Systèmes à compteurs 85
5.2 Systèmes d'addition de vecteurs 86
5.3 Caractérisation des images de Parikh de langages algébriques bornés 91
5.4 Un sous-cas : stratifiabilité des polyèdres d'entiers 94
5.4.1 Réduction au cas des cônes 95
5.4.2 Un critère de bon parenthésage d'un cône décidable en coNP 98
5.5 Algébricité de traces de systèmes à compteurs 101
5.5.1 Algébricité des traces de systèmes à compteurs plats 102
5.5.2 Algébricité des traces de systèmes d'addition de vecteurs 106

6 Conclusion et perspectives **123**
6.1 Autour des arbres de piles 123
6.2 Autour des systèmes à compteurs 126

Chapitre 1

Introduction

J'ai entendu dire un jour que l'informatique consiste à se poser des questions compliquées sur des modèles simples. Le modèle au centre de cette thèse peut illustrer ce propos : un graphe est simplement un ensemble de sommets reliés entre eux par des arcs (orientés ou non) décorés par les lettres d'un alphabet fini. Ce modèle, bien que simple en apparence est pourtant l'un des plus largement étudié en informatique théorique, que ce soit en théorie des graphes, en combinatoire, en vérification, etc... L'une des premières mentions des graphes est due à Euler en 1735. Il considère le problème des sept ponts de Könisberg, qui consiste à déterminer s'il est possible de se promener dans la ville en ne passant qu'une seule fois par chaque pont. Ce problème est résolu négativement en le formalisant comme un graphe, chaque sommet représentant une île ou rive du Pregel, et chaque arc représentant un pont. Ce problème historique est le point de départ de la théorie des graphes, et la généralisation de ce problème à un graphe quelconque, consistant à déterminer l'existence d'un chemin contenant exactement une fois chaque arc est appelé problème du chemin eulérien.

Un graphe peut être utilisé pour représenter une grande variété de systèmes réels, dans le but d'étudier leurs propriétés de manière abstraite. Comme cité plus haut, c'était dès le début l'approche d'Euler qui utilisait un graphe pour représenter les ponts de Könisberg. En informatique, les graphes sont souvent utilisés pour représenter des programmes, le comportement de machines complexes (le système de pilotage d'une fusée, par exemple), ou des systèmes complexes (internet peut aussi être vu comme un graphe). L'idée générale de cette modélisation est par exemple de prendre pour ensemble des sommets les états possibles du système, et de mettre un arc entre deux sommets si il existe une action permettant de passer de l'état représenté par le premier sommet à celui représenté par le second. On veut ensuite étudier ces graphes pour en déduire des propriétés des systèmes qu'ils représentent, par exemple déterminer que le programme représenté ne sera jamais bloqué. On cherchera alors à avoir des preuves formelles de ces propriétés, permettant ainsi que si le graphe étudié représente fidèlement le système étudié, toute propriété prouvée formellement sur le graphe sera vraie sur le système. Par exemple, si on prouve formellement que tout sommet du graphe représentant un programme possède un successeur, alors le programme représenté ne sera jamais bloqué, puisque depuis chacun de ses états (représentés par les sommets du graphe), on a prouvé qu'il existe au moins une action qui peut être effectuée (représentées par les arcs du graphe).

Dans le cas de systèmes relativement simples, un graphe fini sera suffisant pour le modéliser entièrement, même si ce graphe peut être très grand. Dans ce cas, toute propriété sera décidable, étant donné qu'il suffira de la tester pour tous les sommets ou arcs du graphe, ce qui est possible en temps fini. Cela est souvent impraticable, puisque ces graphes peuvent être très grands. Il sera donc désirable de trouver des algorithmes *efficaces*, c'est-à-dire avec une complexité faible, qui permettent de vérifier ces propriétés. C'est le but de l'algorithmique des graphes qui est un

domaine de recherche actif.

Les systèmes plus complexes peuvent en revanche comporter des *variables* pouvant prendre des valeurs non bornées (compteur, horloge, pile, etc). On peut par exemple considérer un programme qui contrôle le comportement d'un ascenseur, que l'on souhaite pouvoir utiliser dans tout ascenseur. Ce programme devra par exemple comporter une variable représentant l'étage courant, qui n'est à priori pas borné. En pratique, ces variables auront une valeur bornée, puisque les systèmes réels ne peuvent pas avoir une mémoire infinie (dans notre exemple, toute tour a un nombre fini d'étages). Cependant, on va garder le caractère non borné des variables dans la modélisation, car :

- en général, la borne n'est pas connue,
- le nombre d'états dans un système borné par une très grande borne peut être extrêmement grand, et cela rendrait les algorithmes sur les graphes finis ineffectifs : même avec une complexité linéaire, appliquer un algorithme à un graphe à 10^{10000} sommets n'est pas praticable, sans parler de la mémoire requise pour stocker un tel graphe.

En gardant des valeurs non bornées pour les variables, on obtient un graphe *infini*, c'est-à-dire avec un nombre infini de sommets et d'arcs. Évidemment, il est impossible de décrire les sommets et les arcs un par un de manière exhaustive, aussi on cherchera à donner des *présentations finies* de ces graphes, c'est-à-dire à décrire leurs sommets et leurs arcs par des définitions d'ensembles et de relations s'exprimant de manière finie. Par exemple, on peut décrire un graphe dont l'ensemble des sommets est l'ensemble des entiers naturels $\mathbb{N}$ et qui contient un arc entre i et $i+1$ pour tout entier i. On décrit ainsi un graphe infini qui peut se représenter comme une demi-droite. Il ne faudra pas perdre de vue que ces graphes infinis ne seront que des abstractions des systèmes réels qu'ils représentent. En effet, en pratique les variables seront bornées, et de plus, dans certaines modélisations, on pourra volontairement ignorer certains phénomènes pour simplifier le modèle obtenu (si on les juge marginaux pour l'étude souhaitée, comme on peut le voir pour le cas des frottements dans certaines études de mécanique). De ce fait, certains chemins existants dans le graphe infini seront absents dans le systèmes réel, par exemple si ils dépassent la borne réelle de certaines variables, ou que les phénomènes ignorés ont une importance dans ce chemin (par exemple, un modèle ignorant les frottements sera très fidèle pour étudier un mouvement dans le vide et le sera beaucoup moins pour étudier le même mouvement dans de la mélasse). Il est donc important, lors de la détermination d'un modèle de bien choisir ce que l'on représente fidèlement et ce que l'on décide d'ignorer pour simplifier le modèle, pour que l'étude menée sur le modèle théorique permette de prédire quelque chose sur le système étudié. En informatique, il est courant d'étudier des propriétés dites *de sûreté*, c'est-à-dire qui interdisent certains comportements (jugés indésirables), pour lesquelles une abstraction consistant à relaxer des conditions est généralement suffisante. En effet, si on a montré, par exemple, qu'une certaine valeur d'une variable est inaccessible dans le graphe infini, en bornant les variables, il sera toujours impossible d'accéder à ces valeurs.

La difficulté de cette approche est que si on considère les graphes infinis dans toutes leur généralité, la plupart des propriétés qu'on peut souhaiter démontrer sont indécidables (et notamment la plupart des propriétés de sûreté citées ci-avant). En effet, le comportement d'une machine de Turing peut être représenté par un graphe infini (les sommets étant les couples formés par un mot du ruban et un état de la machine, et les arcs représentant les transitions de la machine), et il est connu que même des problèmes très simples, comme le problème de l'arrêt, y sont indécidables. Il est donc nécessaire de raffiner l'approche, pour décrire des classes de graphes infinis plus restreintes qui ne contiendront pas les graphes de machine de Turing, et donc pas tous les systèmes possibles, mais sur lesquels on pourra démontrer qu'un certain nombre de propriétés sont décidables, trouver des algorithmes effectifs, et éventuellement les optimiser en terme de complexité. Ces classes permettront entre autres de vérifier des propriétés sur des systèmes réels si on peut trouver un graphe d'une de ces classes qui représente le comportement de ce système.

Cette thèse s'inclut dans ce contexte de déterminer de telles classes de graphes infinis, et de trouver des propriétés qui y sont décidables. Sur les classes que nous définirons, les points

détaillés ci-après retiendront notre attention.

Présentations finies de graphes. Comme annoncé plus haut, quand on a affaire à un graphe infini, il est nécessaire d'en trouver une *présentation finie*. Il existe différentes manières de décrire des graphes, et on classera les présentations que nous utiliserons en deux types :

- les *présentations internes*, qui consisteront à donner explicitement l'ensemble des sommets et l'ensemble des arcs,
- les *présentations externes*, qui consisteront par exemple à décrire un graphe en fonction d'autres graphes, via des transformations, des formules logiques ou des équations.

L'un des points importants dans l'étude de ces présentations est qu'on ne s'intéressera qu'à la *structure* du graphe, en ignorant le nommage des sommets. On ne distinguera donc pas deux graphes qui ne diffèrent que par le nom des sommets, c'est-à-dire tels qu'il existe un *isomorphisme* entre les deux graphes (autrement dit, une bijection entre les sommets des deux graphes telle qu'il existe un arc entre deux sommets du graphe de départ si et seulement si il existe un arc entre leurs images). Deux tels graphes sont dits isomorphes. Toutes les classes qu'on étudiera le seront à isomorphisme près, ce qui permettra de donner plusieurs présentations internes d'un même graphe. L'une des questions principales de notre étude sera donc les différentes présentations internes et externes qu'on peut donner d'une classe de graphes.

Cela permettra notamment de définir des graphes à partir de systèmes réels que l'on souhaite étudier. Il est par exemple courant de modéliser le comportement de programmes complexes comme des graphes dans le but de les étudier.

Accessibilité. Un problème central lorsque l'on étudie des systèmes critiques est de déterminer des *propriétés de sûreté*, c'est-à-dire d'être capable de déterminer si certains comportements indésirables peuvent être observés ou non. Sur le graphe lui-même, cela revient à déterminer si un sommet donné est *accessible* depuis un sommet représentant l'état de départ du système. Ce problème est appelé *problème d'accessibilité*. Dans le cas des machines de Turing, il est indécidable. On prêtera un intérêt particulier à cette question dans les classes de graphes que nous étudierons. Une question liée est de savoir si on peut calculer entièrement l'ensemble des sommets accessibles depuis un état donné. Il n'est pas équivalent au précédent. On verra notamment qu'il existe des cas, comme celui des systèmes d'additions de vecteurs, où le problème d'accessibilité est décidable alors que l'ensemble des états accessibles n'est pas calculable.

Vérification de modèles. L'accessibilité n'est pas le seul problème que l'on peut considérer sur un graphe. Plutôt que d'étudier séparément tous les problèmes auxquels on pourrait vouloir répondre au cas par cas, il existe une manière de décrire formellement des propriétés traitant d'un graphe à l'aide de formules logiques. On va donc classer les problèmes que l'on souhaite déterminer en fonction de la logique dans laquelle ils peuvent s'exprimer. Dans cette étude, nous nous concentrerons sur la logique du premier ordre, la logique du premier ordre augmentée du prédicat d'accessibilité et la logique du second ordre monadique, mais d'autres logiques ont été étudiées sur les graphes (comme le μ-calcul, les logiques temporelles, etc). On considérera le problème de la *vérification de modèles* (aussi connu sous le nom anglais de *model checking*) qui consiste à déterminer si une formule exprimée dans une logique L est satisfaite par un graphe. Un graphe vérifiant une formule ϕ sera dit un *modèle de* ϕ. L'ensemble des formules dans une logique L vérifiées par un graphe sera appelé sa L-théorie. L'un des problèmes nous intéressant sera de déterminer, sur une classe de graphes infinis donnée, pour quelles logiques L la L-théorie de tous les graphes de la classe est décidable, autrement dit de fournir un algorithme qui, étant donné un graphe et une formule de L, détermine si la formule est vérifiée par le graphe.

Langages reconnus. Dans certains cas, on ne souhaite pas étudier le système lui-même, mais plutôt observer ce qu'il produit ou affiche, en voyant les états internes comme une boîte noire.

Dans ce cas, on considère que l'utilisateur ne voit que les étiquettes des arcs empruntés par le calcul dans le graphe, sans information sur les sommets et les arcs réellement empruntés. On va donc regarder les suites d'étiquettes que l'on peut obtenir en partant d'un ensemble de sommets initiaux et en suivant un *chemin* du graphe (c'est-à-dire une suite d'arcs successifs) terminant dans un ensemble de sommets finaux. L'ensemble des suites d'étiquettes ainsi obtenues est appelé le *langage* du graphe (pour ces ensembles initiaux et finaux). On s'intéressera dans la suite aux classes de langages définies par les classes de graphes que nous étudierons, à leurs propriétés, et à leurs relations avec les autres classes de langages connues (comme par exemple ceux de la hiérarchie de Chomsky).

1.1 Historique et contributions

Familles de graphes infinis

Il existe deux résultats fondateurs de l'étude des graphes infinis à isomorphisme près. Le premier de ces résultat est dû à Büchi et Landweber qui ont montré la décidabilité de la MSO-théorie de la demi-droite [BL69], c'est-à-dire l'ensemble $\mathbb{N}$ des entiers muni de la relation successeur. Le second de ces résultats est dû à Rabin, qui a montré la décidabilité de la MSO-théorie de l'arbre binaire complet infini [Rab69], qui peut être vu également comme l'ensemble $\{0,1\}^*$ muni de deux relations successeurs. Cette propriété joue un tel rôle dans les études qui ont suivi qu'elle est appelée théorème de Rabin. Müller et Schupp [MS85] se sont ensuite intéressés aux graphes d'automates à pile, c'est-à-dire les graphes dont les sommets sont les configurations d'un automate à pile accessibles depuis l'état initial, et les arcs représentent l'effet des transitions de l'automate sur ces configurations. Ce travail est le premier où l'existence de plusieurs représentations pour un graphe est abordée. En effet, il considère les graphes connexes obtenus en éliminant tous les sommets atteignables en moins de n pas à partir de la configuration initiale, et montre que l'ensemble des graphes ainsi obtenus, à isomorphisme près, est fini. Cette propriété leur permet de montrer la décidabilité de la MSO-théorie de ces graphes en se servant du théorème de Rabin. Courcelle [Cou90] a étudié les graphes HR-équationnels qui sont les graphes engendrés par des grammaires déterministes de graphes et contiennent strictement les graphes d'automates à pile étudiés par Müller et Schupp. Il a démontré que ces graphes aussi possèdent une MSO-théorie décidable, toujours grâce au théorème de Rabin.

Caucal [Cau92, Cau95, Cau96] a repris l'étude de ces deux classes de graphes, en fournissant une présentation interne des graphes d'automates à pile faisant intervenir des systèmes de réécriture préfixe de mots et une présentation des graphes HR-équationnels comme les graphes obtenus par substitution finie inverse de l'arbre binaire complet, permettant ainsi une démonstration alternative de la décidabilité de la MSO-théorie, puisque les substitutions inverses préservent la décidabilité de la MSO-théorie. Étant donné qu'il les applique à l'arbre binaire complet, le théorème de Rabin est toujours nécessaire à cette preuve. Cela lui a ensuite permis [Cau96] de définir les graphes dits préfixe-reconnaissables [1] qui contiennent strictement les graphes HR-équationnels et possèdent eux-aussi une MSO-théorie décidable. Il en procure une présentation interne consistant en des relations dites préfixe-reconnaissable, qui sont de la forme $(U \times V)W$ avec U, V, W des ensembles reconnaissables de mots, ce qui peut être vu comme une extension de la réécriture préfixe de mots. Stirling [Sti98] a montré que ces graphes sont isomorphes aux graphes des transitions des automates à pile, c'est-à-dire que son ensemble de sommets est un ensemble reconnaissable de configurations et qu'un arc représente l'effet d'une suite de transitions de l'automate contenant une seule transition étiquetée par une lettre et un nombre arbitraire

1. Nous les appellerons suffixe-reconnaissables dans le cadre de cette thèse, car nous en considérerons la version suffixe, alors que la version d'origine utilise la réécriture préfixe. Nous effectuons ce changement car nous considérerons également la réécriture suffixe sur les arbres, pour lesquels les réécritures préfixe et suffixe diffèrent. Par ailleurs, cette appellation n'apparaît pas dans [Cau96], mais dans [CK01], suite à une remarque orale de Niwinsky.

de transitions silencieuses (étiquetées par ε). Barthelman [Bar97] a, quant à lui, fourni une représentation équationnelle de ces graphes en montrant qu'ils sont les solutions des VR-systèmes finis d'équations.

Caucal [Cau02] a ensuite défini la hiérarchie à pile [2], dont le premier niveau est constitué des graphes préfixe-reconnaissables, et le niveau n (dont les éléments sont dits graphes préfixe-reconnaissable d'ordre n) est obtenu en appliquant une opération de structure arborescente (introduite par Shelah [She75]) suivie d'une substitution rationnelle inverse à un graphe du niveau $n-1$. Il est également possible de considérer que le niveau 0 de la hiérarchie est constitué des graphes finis. L'opération de structure arborescente préservant la décidabilité de la MSO-théorie [Wal96], il s'ensuit que tous les graphes appartenant à la hiérarchie préfixe-reconnaissable disposent d'une MSO-théorie décidable. Carayol et Wöhrle [CW03] ont montré que les graphes préfixe-reconnaissables d'ordre n sont les graphes des transitions des automates à n niveaux de pile (définis de manière analogue à ceux des automates à pile). Carayol et Colcombet [CC03] ont montré que ces graphes sont également les solutions des VR-systèmes d'équations exprimés sur un graphe de niveau $n-1$. Carayol [Car05] a également montré que ces graphes peuvent aussi être vus comme ayant des ensembles reconnaissables de piles d'ordre n pour ensemble de sommets et dont les arcs sont définis par des relations préfixe-reconnaissables d'ordre n. Pour définir les ensembles reconnaissables de piles d'ordre n, il utilise une notion de reconnaissabilité sur les suites d'opérations sur des piles d'ordre n. Fratani [Fra05] a montré que les ensembles reconnaissables de piles d'ordre n définis par Carayol coïncident avec les ensembles de piles d'ordre n définis par des MSO-formules.

Dans une optique différente, Khoussainov et Nérode [KN95], ont défini les graphes automatiques dont les sommets sont des mots et les arcs représentent des relations reconnaissables (c'est-à-dire acceptées par des transducteurs de mots) qui disposent d'une FO-théorie décidable [BG00]. En remarquant que les relations arbre-synchronisées définies par Dauchet et Tison [DT90] contiennent strictement les relations reconnaissables, Blumensath et Grädel [Blu99, BG00] ont étendu la notion de graphes automatiques aux graphes arbre-automatiques dont les sommets sont des arbres et les arcs représentent des relations arbre-synchronisées (c'est-à-dire acceptées par des transducteurs d'arbres). Ces graphes disposent également d'une représentation équationnelle : ce sont les solutions des VRS-systèmes d'équations finis [Col04]. Ils disposent aussi d'une représentation par transformations : ils sont obtenus par une interprétation à ensembles finis à partir du dépliage d'un graphe fini [CL07], donnant ainsi une preuve alternative de la décidabilité de la FO-théorie de ces graphes. Toujours dans [CL07], cette dernière représentation est utilisée pour définir la hiérarchie arbre-automatique : un graphe arbre-automatique d'ordre n étant obtenu par une interprétation à ensembles finis du dépliage d'un graphe préfixe-reconnaissable d'ordre $n-1$, et ayant donc également une FO-théorie décidable, et n'ayant pas de représentation interne connue. Toutefois, la représentation équationnelle des graphes arbre-automatiques présente dans [Col04] s'étend à la hiérarchie arbre-automatique : un graphe arbre-automatique d'ordre n est la solution d'un VRS-système d'équations exprimé sur un graphe préfixe-reconnaissable d'ordre $n-1$. Par ailleurs, cette hiérarchie est stricte [CL07].

Il existe une classe de graphes contenant strictement les graphes préfixe-reconnaissables et strictement contenue dans les graphes arbre-automatiques : les graphes des systèmes de réécriture suffixe d'arbres. Ils disposent d'une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable [DT90], résultat obtenu en introduisant la notion de transducteurs suffixes d'arbres qui contiennent la clôture par itération des systèmes de réécriture suffixe d'arbres et sont eux-mêmes clos par itération, et en montrant que leurs graphes sont arbre-automatiques. Ces graphes sont également les solutions des VRA-systèmes d'équations finis [CC03].

Ces classes de graphes, et leurs inclusions, sont schématisées à la figure 1.1. On observera l'absence de graphes situés entre les niveaux n des hiérarchies préfixe-reconnaissable et arbre-automatique disposant d'une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable.

2. qui n'est pas éponyme.

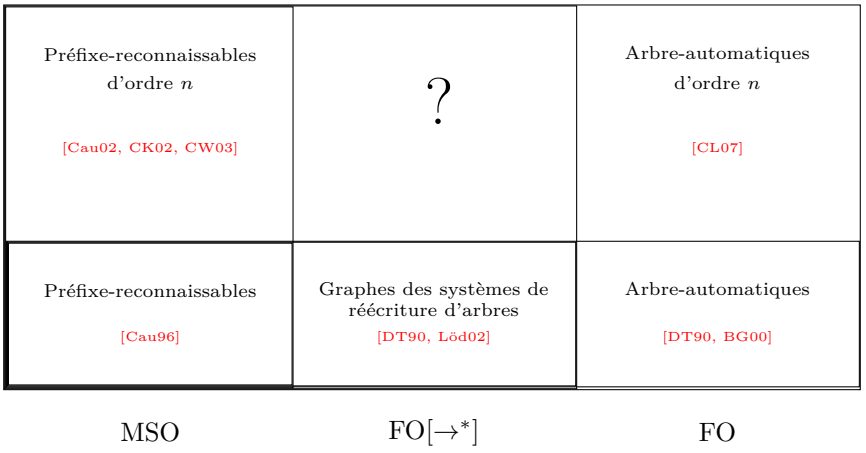


FIGURE 1.1 – Inclusions des classes de graphes décrites dans cette introduction.

Systèmes de réécriture suffixe d'arbres de piles

La première contribution de ce document consiste à proposer un modèle donnant une représentation interne de graphes ayant ces propriétés. Dans ce but, nous présentons les arbres de piles d'ordre n, qui sont des arbres dont les nœuds sont étiquetés par des piles d'ordre $n-1$, avec la convention qu'une pile d'ordre 0 est une lettre. Cette structure de donnée est la plus simple contenant à la fois les piles d'ordre n (qui peuvent être vus comme des arbres de piles d'ordre n unaires) et les arbres (qui sont des arbres de piles d'ordre 1). Nous y associons une notion d'opération qui étend à la fois les opérations de piles des automates à pile d'ordre n et les opérations de réécriture suffixe d'arbres, qui permet de définir des systèmes de réécriture suffixe d'arbres de piles, définissant ainsi des graphes qui contiennent à la fois les graphes préfixe-reconnaissables d'ordre n et les graphes des systèmes de réécriture d'arbres. On introduit ensuite une notion de reconnaissabilité sur les opérations de réécriture suffixe d'arbres de piles, et une notion liée de transducteurs suffixes d'arbres de piles qui permet à la fois d'étendre les relations de transitions des automates à piles et les transducteurs suffixes d'arbres, et qui contient les clôtures par itérations des relations de réécriture suffixe d'arbres. Enfin, on montre à l'aide d'une interprétation à ensembles finis depuis un graphe préfixe-reconnaissable d'ordre n que les graphes des systèmes de réécriture d'arbres de piles d'ordre n et des transducteurs suffixes d'arbres de piles d'ordre n sont des graphes arbre-automatiques d'ordre n. Tout cela permet de montrer qu'ils disposent d'une $\mathsf{FO}[\overset{*}{\rightarrow}]$-théorie décidable, et qu'ils permettent bien de combler le trou de la figure 1.1. De plus, la notion de systèmes de réécriture suffixe d'arbres de piles est la plus simple permettant de définir des graphes ayant une $\mathsf{FO}[\overset{*}{\rightarrow}]$-théorie décidable et étendant à la fois les graphes de la hiérarchie à pile et ceux des systèmes de réécriture suffixe d'arbres. Nous présentons également un exemple de langage reconnu par un système de réécriture suffixe d'arbres de piles dont nous conjecturons qu'il n'est reconnu ni par un automate à piles d'ordre supérieur ni par un système de réécriture suffixes d'arbres. Cette contribution a fait l'objet d'une publication à CSR2015 [Pen15] et est présentée au chapitre 4.

Systèmes à compteurs

Une autre approche consiste à partir de modèles Turing-complets et de les restreindre pour récupérer la décidabilité de certaines propriétés. L'un des modèles Turing-complets les plus largement étudiés est celui des systèmes à compteurs, qui disposent de plusieurs formalismes équivalents différents, comme celui des machines de Minsky [Min67]. Un système à compteurs peut être vu comme un automate à mémoire, dont la mémoire est constituée de plusieurs compteurs d'entiers, et qui peut effectuer des actions modifiant son état de contrôle, la valeur de ses

compteurs et qui peut tester la valeur de ces compteurs. L'une des restrictions des systèmes à compteurs la plus anciennement étudiée est celle des systèmes d'addition de vecteurs introduits par Karp et Miller [KM69] (ainsi que leur version avec états de contrôle [HP79]), qui revient à autoriser seulement des modifications de compteurs qui sont des additions de constantes aux compteurs, sans possibilité de tester leur valeur. La popularité de ce modèle provient entre autre du fait qu'il est équivalent aux réseaux de Petri [Pet62] qui sont très utilisés pour la modélisation de systèmes réels, notamment ceux faisant intervenir des actions pouvant s'effectuer de manière indépendante les unes des autres. Par ailleurs, un grand nombre de problèmes indécidables sur les systèmes à compteurs deviennent décidables dans le cas des systèmes d'additions de vecteurs, comme le problème de couverture [KM69] (revenant à déterminer si on peut atteindre un vecteur plus grand qu'un vecteur donné) et le problème d'accessibilité d'un vecteur [ST77, May81, Kos82, Lam92, Ler10]. Cependant, on ne peut en général pas déterminer l'ensemble des vecteurs accessibles à partir d'un vecteur donné (résultat attribué à Rabin dans [KM69]). Les propriétés structurelles des graphes associés à des systèmes d'addition de vecteurs ont été peu étudiées, mais notons toutefois que la décidabilité de la FO-théorie de différentes variantes de ces graphes a été étudiée dans [DDMM11]. Une autre thématique développée autour des systèmes d'additions de vecteurs est celle de leurs langages. Il a notamment été montré qu'il est possible de déterminer si les langages de systèmes d'additions de vecteurs sont reconnaissables [GY80, VVN81] ou algébriques [Sch92]. Cependant, ces preuves ne sont valables que si on considère les langages des traces des systèmes d'addition de vecteurs, c'est-à-dire les langages d'actions (autrement dit, chaque lettre n'est associée qu'à une seule action). Dans le cas contraire, ces problèmes sont indécidables. Cela permet de déterminer l'existence de modèles plus simples (automates ou automates à pile) ayant le même comportement observable, quand bien même le comportement interne du système d'addition de vecteurs est plus compliqué que celui d'un automate ou d'un automate à pile.

L'étude de cette dernière propriété repose sur le graphe de couverture des systèmes d'addition de vecteurs dit graphe de Karp et Miller [KM69], et sur la caractérisation des langages algébriques bornés introduite par Ginsburg [GS64] qui dit qu'un langage borné est algébrique si et seulement si son image de Parikh est un ensemble semi-linéaire stratifiable. Cette caractérisation est un raffinement de l'étude de Parikh sur les langages algébriques [Par61, Par66] qui dit que si un langage est algébrique, alors son image de Parikh est un ensemble semi-linéaire stratifiable. Cependant, déterminer si un ensemble semi-linéaire est stratifiable reste un problème ouvert à ce jour. Les ensembles semi-linéaires sont également reliés à l'arithmétique de Presburger [GS66] et ont été exploités pour étudier les systèmes d'addition de vecteurs [Ler10, Ler13c, Ler13b, Ler13a].

L'autre restriction des systèmes à compteurs présente dans la littérature consiste à se ramener à des systèmes à compteurs plats, c'est-à-dire dont les langages d'actions forment justement des langages bornés. Cette approche a été utilisée dans [AG68, BFLP08, BW94, BH99, BIK10, CJ98, FIS03], et permet notamment de décider des propriétés d'accessibilité similaires à celles des systèmes d'addition de vecteurs.

Algébricité des systèmes à compteurs

La seconde contribution de ce document commence par étudier la stratifiabilité des ensembles semi-linéaires. Nous laissons ouvert le problème général posé par Ginsburg et Spanier, mais nous montrons que si on se restreint au sous-cas des polyèdres d'entiers, ce problème est décidable, et est de plus coNP-complet. En nous servant de la caractérisation des langages algébriques bornés de Ginsburg et Spanier, nous appliquons ce résultat aux systèmes à compteurs plats en montrant que les images de Parikh des langages des traces des systèmes à compteurs plats sont des polyèdres d'entiers, et en déduisons la coNP-complétude du problème d'algébricité des traces de systèmes à compteurs plats. Ce travail a été publié avec Jérôme Leroux et Grégoire Sutre à ATVA2014 [LPS14]. Un autre problème ayant retenu notre attention est celui de l'algébricité des traces de systèmes d'addition de vecteurs. En effet, la preuve proposée par Schwer [Sch92] contient des inexactitudes, est loin d'être simple et ne procure pas d'indice sur la complexité de

ce problème, bien que la piste choisie repose sur des idées intéressantes et non triviales. Nous proposons une nouvelle preuve de la décidabilité de ce problème qui ne repose plus sur l'arbre de Karp et Miller, mais utilise toujours la caractérisation des langages algébriques bornés par des ensembles semi-linéaires stratifiables. Cette preuve repose sur la construction d'un algorithme qui construit un automate à pile qui reconnaît le langage des traces du système d'addition de vecteurs si ce dernier est algébrique, et permet de trouver un langage borné non algébrique inclus dans le langage du système d'addition de vecteurs dans le cas contraire (montrant ainsi la non-algébricité du langage de ce système, puisque c'est l'intersection de ce langage avec un langage reconnaissable, et que l'intersection d'un langage algébrique et d'un langage reconnaissable est toujours algébrique). Ce résultat a fait l'objet d'une publication conjointe avec Jérôme Leroux et Grégoire Sutre à LICS2013 [LPS13a]. Leroux, Praveen et Sutre ont ensuite montré que ce problème est ExpSpace-complet [LPS13b].

1.2 Plan de la thèse

Dans le chapitre 2, nous définissons les notions de base utilisées dans ce document, en particulier, les notions de relations, de graphes et de logiques y seront rappelées.

Le chapitre 3 effectue un état de l'art des connaissances sur les graphes infinis, en insistant sur des classes qui jouent un rôle dans nos contributions. Dans un premier paragraphe, nous présentons plus précisément les différentes manières de décrire un graphe et les questions qui nous intéresseront dans la suite du document : équivalence des représentations, problème d'accessibilité, problème de vérification de modèles et langages définis. Le reste du chapitre est consacré à la présentation de classes de graphes déjà connues. Le deuxième paragraphe s'intéresse à la hiérarchie à pile, que nous choisissons de renommer hiérarchie suffixe-reconnaissable. On rappellera son lien avec les automates à pile et les automates à pile d'ordre supérieur, la décidabilité de sa MSO-théorie et que les langages reconnus par ces graphes sont ceux de la hiérarchie OI. Enfin, on présentera une présentation interne alternative de ces graphes : les systèmes de réécriture de piles d'ordre supérieur. L'intérêt de ce formalisme sera de lier plus facilement ces graphes aux autres classes étudiées dans le chapitre 4. Le troisième paragraphe sera consacré aux graphes définis par des systèmes de réécriture suffixe d'arbre, en insistant sur les différentes représentations connues de ces graphes, leurs propriétés de décidabilité logique et les langages qu'ils reconnaissent. On notera qu'ils contiennent le niveau 1 de la hiérarchie suffixe-reconnaissable et qu'ils possèdent une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable. Enfin, nous parlerons de la hiérarchie arbre-automatique, qui est une classe de graphes moins connue que les précédentes, les seules choses connues à son sujet étant la décidabilité de la FO-théorie des graphes qu'elle contient, et le fait qu'elle soit stricte. Le dernier paragraphe récapitulera les relations entre ces classes de graphes, notamment à l'aide d'une représentation graphique. Cela permettra notamment de noter une lacune dans les connaissances sur les classes de graphes : l'absence d'une classe de graphes contenant à la fois le niveau n de la hiérarchie suffixe-reconnaissable et les graphes de systèmes de réécriture suffixe d'arbres possédant une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable.

Le chapitre 4 présente les systèmes de réécriture suffixe d'arbres de piles, dont nous avons parlé plus haut, qui étendent à la fois les notions d'automate à pile d'ordre supérieur et de système de réécriture d'arbres. Ce modèle induit une classe de graphes possédant des propriétés similaires à celles liées aux systèmes de réécriture d'arbres. Le premier paragraphe est consacré à la présentation formelle du modèle, en insistant sur la définition des opérations qu'on autorise. Le second paragraphe est consacré à prouver que ce modèle est une extension des systèmes de réécriture suffixe d'arbres et des systèmes de réécriture de piles. Le troisième paragraphe définit une notion de reconnaissabilité sur les opérations grâce à des automates d'opérations, en déduit une notion de reconnaissabilité sur les arbres de piles et montre que les automates d'opérations admettent une forme normalisée. Le quatrième paragraphe s'intéresse aux classes de graphes liées aux systèmes de réécriture suffixe d'arbres de piles et montre que les graphes de ces classes disposent d'une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable (en montrant qu'ils sont inclus dans la hiérarchie arbre-

automatique). Enfin, le dernier paragraphe s'intéresse aux langages qu'on peut définir sur ces classes, en en fournissant un exemple et en présentant quelques conjectures sur ces langages, n'ayant malheureusement que peu de résultats concrets sur ce point pour l'instant.

Le chapitre 5 s'intéresse aux systèmes à compteurs et aux systèmes d'additions de vecteurs, et notamment aux langages des traces de ces systèmes, à travers le problème consistant à déterminer si ces langages sont algébriques. Le premier paragraphe définit les systèmes à compteurs. Le second définit les systèmes d'addition de vecteurs, et rappelle l'état de l'art des connaissances sur ce modèle. Le troisième paragraphe rappelle que le problème consistant à décider si un langage borné est algébrique est équivalent à un problème décidant la stratifiabilité des ensembles semi-linéaires, grâce à la caractérisation de ces langages par leur image de Parikh. Le quatrième paragraphe montre que si on se restreint au cas des polyèdres d'entiers, le problème de stratifiabilité d'un ensemble semi-linéaire est décidable. Le cinquième paragraphe applique ce résultat pour montrer que l'algébricité d'un système à compteurs plats est décidable, puis montre qu'on peut également décider l'algébricité d'un système d'addition de vecteurs général, en donnant à chaque fois des algorithmes effectifs.

Chapitre 2

Préliminaires

2.1 Notions de base

2.1.1 Ensembles et sous-ensembles

On note $\emptyset$ l'ensemble vide, $\mathbb{N}$ l'ensemble des entiers naturels, $\mathbb{Z}$ l'ensemble des entiers et $\mathbb{Q}$ l'ensemble des nombres rationnels. Étant donnés deux ensembles Γ et Γ', Γ' est un *sous-ensemble* de Γ si $\forall x, x \in \Gamma' \Rightarrow x \in \Gamma$. On note $\Gamma' \subseteq \Gamma$. On note $\mathcal{P}(\Gamma)$ l'ensemble des sous-ensembles de Γ, appelé aussi *ensemble des parties* de Γ. $\mathcal{P}_f(\Gamma)$ est l'ensemble des parties finies de Γ. Étant donnés deux ensembles Γ et Γ', on définit :

- leur *union* $\Gamma \cup \Gamma' = \{x \mid x \in \Gamma \vee x \in \Gamma'\}$;
- leur *intersection* $\Gamma \cap \Gamma' = \{x \mid x \in \Gamma \wedge x \in \Gamma'\}$;
- leur *produit* $\Gamma \times \Gamma' = \{(x, y) \mid x \in \Gamma, y \in \Gamma'\}$;
- l'ensemble Γ^n des *suites de taille n d'éléments* de Γ :
  - $\Gamma^0 = \{\varepsilon\}$, où ε est un élément spécial appelé «suite vide» ;
  - $\Gamma^n = \Gamma \times \Gamma^{n-1}$, pour $n \geq 1$.
- l'ensemble Γ^* des suites finies d'éléments de Γ, $\Gamma^* = \bigcup_{n \in \mathbb{N}} \Gamma^n$;
- l'ensemble Γ^* des suites finies non vides d'éléments de Γ, $\Gamma^+ = \bigcup_{n>0} \Gamma^n$;
- le *complémentaire* de Γ dans Γ', $\overline{\Gamma}^{\Gamma'} = \{x \in \Gamma' \mid x \notin \Gamma\}$ si $\Gamma \subseteq \Gamma'$. Quand Γ' est sous-entendu, on note simplement $\overline{\Gamma}$;
- le *cardinal* d'un ensemble fini Γ est son nombre d'éléments. Il est noté $|\Gamma|$.

2.1.2 Relations

Une *relation* R entre n ensembles $\Gamma_1, \cdots, \Gamma_n$ est un sous-ensemble de $\Gamma_1 \times \cdots \times \Gamma_n$. On appelle n l'*arité* de la relation. On parle de relation n-aire.

Étant donné un élément $(x_1, \ldots, x_n)$ d'une relation R, on note également $R(x_1, \ldots, x_n)$.

Étant donnée une relation sur Γ^n, sa *projection* sur les coordonnées $i_1, \ldots, i_k$ (où les i_j sont des entiers inférieurs à n, tous différents) est la relation sur Γ^k définie comme suit :

$$\pi_{i_1,\ldots,i_k}(R) = \{(\alpha_{i_1}, \ldots, \alpha_{i_k}) \mid (\alpha_1, \ldots, \alpha_n) \in R\}.$$

Étant donnée une relation sur Γ^n, sa *cylindrification* est la relation sur Γ^{n+1} définie comme suit :

$$\mathrm{cyl}(R) = \{(\alpha_1, \ldots, \alpha_{n+1}) \mid (\alpha_1, \ldots, \alpha_n) \in R \wedge \alpha_{n+1} \in \Gamma\}.$$

Si $n = 2$, on parle d'une *relation binaire*. Si $n = 2$ et $\Gamma_1 = \Gamma_2 = \Gamma$, on dit que R est une relation binaire sur Γ. Une relation binaire R sur Γ est dite :

- *réflexive* si $\forall x \in \Gamma,\ R(x, x)$;
- *symétrique* si $\forall x, y \in \Gamma,\ R(x, y) \iff R(y, x)$;
- *antisymétrique* si $\forall x, y \in \Gamma,\ R(x, y) \wedge R(y, x) \Rightarrow x = y$;
- *transitive* si $\forall x, y, z \in \Gamma,\ R(x, y) \wedge R(y, z) \Rightarrow R(x, z)$.

On note id_Γ la *relation identité* de Γ : $\mathrm{id}_\Gamma = \{(x, x) \mid x \in \Gamma\}$. Quand Γ est sous-entendu, on note simplement id.

Étant données R et R' deux relations sur Γ, on définit leur *composition*

$$R \circ R' = \{(x, y) \mid \exists z \in \Gamma, (x, z) \in R \wedge (z, y) \in R'\}.$$

Étant donnée une relation R sur Γ et un entier k, on note R^k la relation définie récursivement par :

- $R^0 = \mathrm{id}$.
- $\forall x, y \in \Gamma, R^k(x, y) \iff \exists z \in \Gamma, R(x, z) \wedge R^{k-1}(z, y)$.

On note $R^* = \bigcup_{n \in \mathbb{N}} R^n$ la *fermeture réflexive et transitive* de R.

On note $R^{-1} = \{(x, y) \mid (y, x) \in R\}$ la *relation inverse* de R.

Une relation est dite *d'équivalence* si elle est *réflexive, symétrique* et *transitive*.

Une relation est dite *d'ordre* si elle est *réflexive, antisymétrique* et *transitive*. Elle est dite *totale* si de plus, $\forall x, y \in \Gamma, R(x, y) \vee R(y, x)$.

Une relation binaire R sur $\{1, \ldots, d\}$ est dite *bien parenthésée* si

$$\forall i, j \in \{1, \ldots, d\}, (i, j) \in R \Rightarrow i \leq j,$$

et

$$\forall i, j, k, l \in \{1, \ldots, d\}, (i, j) \in R \wedge (l, k) \in R \Rightarrow \neg(i < k < j < l).$$

Informellement, si on représente une relation bien parenthésée comme des arcs situés au dessus de la droite des entiers, aucun arc ne se croise.

2.1.3 Fonctions

Une *fonction* f d'un ensemble Γ vers Γ' est une relation binaire entre Γ et Γ' telle que $\forall x \in \Gamma, y, y' \in \Gamma',\ (x, y) \in f \wedge (x, y') \in f \Rightarrow y = y'$. On note alors $f(x)$ l'unique y tel que $(x, y) \in f$, et on appelle y *l'image* de x par f et x *un antécédent* de y par f.

On appelle *ensemble de définition* de f l'ensemble $\mathrm{Def}(f) = \{x \in \Gamma \mid \exists y \in \Gamma', y = f(x)\}$.

On appelle *image* de f l'ensemble $\mathrm{Im}(f) = \{y \in \Gamma' \mid \exists x \in \Gamma, y = f(x)\}$.

Une fonction f de Γ dans Γ' est appelée une *application* si $\mathrm{Def}(f) = \Gamma$.

Une fonction f de Γ dans Γ' est dite :

- *injective* si $\forall x, x' \in \Gamma,\ f(x) = f(x') \Rightarrow x = x'$;
- *surjective* si $\mathrm{Im}(f) = \Gamma'$;
- *bijective* si elle est à la fois injective et surjective.

Étant donnée une fonction injective f de Γ dans Γ', on appelle *l'inverse* de f, notée f^{-1} l'unique fonction de Γ' dans Γ telle que $\forall x \in \Gamma, y \in \Gamma', f^{-1}(y) = x$ si et seulement si $f(x) = y$.

2.1.4 Morphismes

Un *morphisme* ϕ d'un ensemble Γ dans un ensemble Γ' pour les relations $r_1, \cdots, r_k$ est une fonction de Γ dans Γ' telle que pour tout i, et $x, y \in \Gamma$, $r_i(x, y) \Leftrightarrow r_i(\phi(x), \phi(y))$. Si de plus ϕ est bijectif, on dit que c'est un isomorphisme.

2.1.5 Vecteurs de nombres

Pour les ensembles $\mathbb{N}$, $\mathbb{Z}$ et $\mathbb{Q}$, on parlera de *vecteur* pour désigner les suites finies d'éléments de ces ensembles. Étant donné $\vec{v} \in \mathbb{Q}^n$ (resp $\mathbb{N}^n$, $\mathbb{Q}^n$), n est appelé la *dimension* de v. On définit $\|\vec{v}\| = \{i \in \{1, \cdots, n\} \mid \vec{v}(i) \neq 0\}$ le *support* de $\vec{v}$, $\|\vec{v}\|^+ = \{i \in \{1, \cdots, n\} \mid \vec{v}(i) > 0\}$ son *support positif* et $\|\vec{v}\|^- = \{i \in \{1, \cdots, n\} \mid \vec{v}(i) < 0\}$ son support négatif. On définit aussi un ordre partiel sur les vecteurs de nombres : étant donnés deux vecteurs $\vec{v}$ et $\vec{v'}$ de dimension n, on a $\vec{v} \geq \vec{v'}$ si et seulement si pour tout $i \leq n$ on a $\vec{v}(i) \geq \vec{v'}(i)$. On a de plus $\vec{v} > \vec{v'}$ si $\vec{v} \geq \vec{v'}$ et $\exists i \leq n$ tel que $\vec{v}(i) > \vec{v'}(i)$.

2.1.6 Matrices

Une matrice de taille $m \times n$ à valeurs dans un ensemble Γ est une application de $[1, m] \times [1, n]$ dans Γ. Graphiquement, une matrice est représentée comme un tableau.

Dans ce document, on utilisera essentiellement des matrices à valeurs dans $\mathbb{Z}$.

Étant donné un vecteur $\vec{v}$ de dimension n et une matrice $\mathbf{A}$ de dimension $m \times n$, on définit le produit de $\mathbf{A}$ par $\vec{v}$ comme étant le vecteur $\vec{v'}$ de dimension m, noté $\vec{v'} = \mathbf{A}\vec{v}$, tel que pour tout $i \leq m$, $\vec{v'}(i) = \sum_{1 \leq j \leq n} (\mathbf{A}(i, j) \times \vec{v}(i))$.

Dans ce document, on utilisera les matrices pour représenter un ensemble d'équations ou contraintes sur un même ensemble de variables (représenté par un vecteur). L'ensemble de contraintes sera alors représenté par une contrainte sur le produit d'un vecteur et d'une matrice.

Exemple 2.1. *L'ensemble des nombres x_1, x_2, x_3 tels que $3x_1 + 2x_2 \geq 4$ et $2x_1 - 6x_3 \geq 7$ est exactement l'ensemble vérifiant que*

$$\begin{pmatrix} 3 & 2 & 0 \\ 2 & 0 & -6 \end{pmatrix} (x_1, x_2, x_3) \geq (4, 7)$$

2.2 Mots et langages

Étant donné un ensemble Σ, qu'on appelle *alphabet*, un mot u sur Σ est une suite finie d'éléments de Σ. On note $u = u_1 \cdots u_{|u|}$, où $|u|$ est la *taille* de u. Le mot vide est la suite de taille 0, notée ε. Alternativement, on peut voir un mot u comme une fonction de $\mathbb{N}\backslash\{0\}$ dans Σ telle que $dom(u)$ est fini et clos par $\leq$, c'est-à-dire que $dom(u) = [1, |u|]$. L'ensemble des mots sur Σ est Σ^*. On appelle *langage* sur Σ un sous-ensemble de Σ^*. Étant donnés deux mots $u = u_1 \cdots u_m$ et $v = v_1 \cdots v_n$, leur *concaténation* $u.v$ est le mot $u_1 \cdots u_m v_1 \cdots v_n$. Étant donnés deux langages L et L', on définit leur *concaténation* $L.L' = \{u.v \mid u \in L, v \in L'\}$. Par abus de notation, on note $u.L = \{u\}.L$ et $L.u = L.\{u\}$.

L'ensemble Rat des langages rationnel est la clôture des langages finis par union, concaténation et itération de la concaténation.

Étant donnés deux mots $u = u_1 \cdots u_m$ et $v = v_1 \cdots v_n$, on dit que u est un *préfixe* de v si $m \leq n$ et $\forall i \leq m, u_i = v_i$. On note alors $u \sqsubseteq v$. On dit que u est un *suffixe* de v si $m \leq n$ et $\forall i \leq m, u_i = v_{n-m+i}$.

Un langage L est dit *clos par préfixe* si $u \in L \Rightarrow (\forall v, v \sqsubseteq u \Rightarrow v \in L)$.

Un langage L est dit *borné* si il existe $u_1, \cdots, u_k \in \Sigma^*$ tels que $L \subseteq u_1^* \cdots u_k^*$. Cette notion a été introduite par Ginsburg et Spanier [GS64] sous le nom anglais de *bounded language*, le terme borné en étant une traduction littérale.

2.3 Graphes

Un graphe étiqueté par un ensemble Σ est un couple $\mathcal{G} = (V, E)$ tel que :

· l'ensemble des sommets V est un ensemble quelconque.

· l'ensemble des arcs E est un sous-ensemble de $V \times \Sigma \times V$.

Étant donnés deux sommets x, y de $\mathcal{G}$, on note :

· $x \xrightarrow[G]{a} y$ si $(x, a, y) \in E$.

· $x \xrightarrow[G]{} y$ si $\exists a \in \Sigma, (x, a, y) \in E$.

· $x \xrightarrow[G]{}^{-1} y$ si $y \xrightarrow[G]{} x$.

· Étant donné $v \in \Sigma^*$, on définit récursivement $x \xrightarrow[G]{v} y$.

  - $x \xrightarrow{\varepsilon} y$ si et seulement si $x = y$;
  - $x \xrightarrow{v} y$, avec $v = a.v'$ si et seulement si $\exists z \in V, x \xrightarrow[G]{a} z \wedge z \xrightarrow[G]{v'} y$.

· $x \xrightarrow[G]{*} y$ si il existe un *chemin* de x à y, c'est-à-dire $\exists v \in \Sigma^*, x \xrightarrow[G]{v} y$; on appelle v l'*étiquette* du chemin.

· $x \xrightarrow[G]{+} y$ si il existe un chemin non trivial de x à y, c'est-à-dire $\exists v \in \Sigma^+, x \xrightarrow[G]{v} y$.

Dans tout ce qui précède, on omet G quand il n'y a pas ambiguïté.
Un graphe $\mathcal{G} = (V, E)$ est dit :

· *fini* si V est fini.

· *connexe* si $\forall x, y \in V, x(\rightarrow \cup \rightarrow^{-1})^* y$.

· *sans circuit* si $\mathcal{G}$ ne contient pas de circuit, c.à.d. $\forall x \in V, x \xrightarrow{+} x$.

· *déterministe* si $\forall x, y, z \in V, \gamma \in \Sigma, x \xrightarrow{\gamma} y \wedge x \xrightarrow{\gamma} z \Rightarrow y = z$.

Un graphe sans circuit est appelé un *DAG* (pour *directed acyclic graph*). Un graphe sans circuit connexe $\mathcal{G}$ qui contient un sommet r tel que, $\forall y \in V$ il existe un unique chemin de r à y est appelé un *arbre*. Le sommet r est appelé la *racine* de $\mathcal{G}$.

Étant donnés un graphe $\mathcal{G}$ et deux ensembles quelconques de sommets I et F, on appelle *langage des traces* ou simplement *langage* de $\mathcal{G}$ entre I et F l'ensemble des étiquettes des chemins commençant par un sommet de I et terminant par un sommet de F. Formellement, on a $\mathcal{L}(\mathcal{G}, I, F) = \{u \mid \exists x \in I, y \in F, x \xrightarrow{u} y\}$. Dans certains cas, on pourra considérer des arcs étiquetés par le mot vide ε, qui n'auront pas d'influence sur l'étiquette des chemins qui les empruntent. C'est par exemple le cas des ε-transitions des automates finis. Dans la suite, sauf mention contraire, les graphes que nous considéreront ne contiendront pas de tels arcs. Dans la plupart des cas, sur les graphes infinis, les classes de langages étudiées seront restreintes aux langages entre deux ensembles finis de sommets (éventuellement étendus à certains cas particuliers d'ensembles infinis).

2.4 Automates finis et langages reconnaissables

Un automate fini est un graphe fini, auquel on adjoint deux ensembles de sommets pour y associer un unique langage. On donne ici la définition classique d'un automate fini.

Définition 2.1. *Un automate fini $\mathcal{A}$ est un quintuplet $(Q, \Sigma, I, F, \Delta)$ tel que :*

· *Σ est un ensemble fini appelé alphabet.*

· *Q est un ensemble fini d'états.*

· *$I \subseteq Q$ est l'ensemble des* états initiaux*.*

· *$F \subseteq Q$ est l'ensemble des* états finaux*.*

· *$\Delta \subseteq Q \times \Sigma \times Q$ est l'ensemble des transitions.*

On peut observer que cette définition est bien équivalente à une présentation de l'automate $\mathcal{A}$ comme le graphe (Q, Δ), étiqueté par Σ, auquel on associe les deux sous-ensembles I et F de Q. Le langage reconnu par l'automate $\mathcal{A}$, $\mathcal{L}(\mathcal{A})$ est le langage $\mathcal{L}((Q, \Delta), I, F)$.

Un langage est dit *reconnaissable* si il est reconnu par un automate fini. Les langages rationnels sont exactement les langages reconnaissables.

2.5 Logiques

En général, une *logique* est un langage mathématique permettant d'exprimer des propriétés sur des objets tels que les mots, les graphes, ou – plus généralement – des *structures relationnelles*. Formellement, une logique contient des *formules atomiques* qui expriment des propriétés «de base» de la structure qui sont liés par des connecteurs (qui sont des fonctions booléennes) et peuvent posséder des *quantificateurs* sur des variables. Dans la suite, on considérera essentiellement des logiques exprimant des propriétés sur des graphes, étant donné que c'est cet objet qui nous intéresse dans le cadre de cette thèse. Cependant, dans la deuxième partie, on considérera des logiques n'exprimant pas des propriétés sur des graphes, on définit donc les logiques dans un cadre général.

2.5.1 Structures relationnelles

On présente ici les notions de signature et de structure relationnelle, qui permettent de définir les logiques dans un cadre général.

Définition 2.2. *Une* signature *est un ensemble fini $\mathcal{R}$ de symboles relationnels, chaque symbole R possédant une arité entière notée $|R|$.*

Une structure relationnelle *$\mathcal{S}$ de signature $\mathcal{R}$ est un couple $(\mathcal{U}, \iota)$ où $\mathcal{U}$ est un ensemble quelconque appelé* univers de la structure *et ι est une application appelée* interprétation *associant à chaque symbole R de $\mathcal{R}$, une relation $\iota(R)$ sur $\mathcal{U}^{|R|}$.*

La notion de signature séparée de celle de structure est utile dans les définitions. Cependant, par soucis de simplicité, dans la suite de ce document, on omettra la plupart du temps la notion de signature, en présentant simplement les structures comme des uplets $(\mathcal{U}, R_1, \cdots, R_k)$, avec $R_1, \cdots, R_k$ des relations sur $\mathcal{U}$, la signature étant implicitement $\{R_1, \cdots, R_k\}$ et l'interprétation ι étant laissée implicite.

Notons que nous n'avons pas introduit de symboles de constantes. Cela est dû au fait qu'un symbole unaire interprété par un singleton peut être vu comme une constante de la structure.

La plupart des objets mathématiques usuels peuvent se voir comme des structures relationnelles. Par exemple, le corps des complexes peut se voir comme une structure de signature $\{0, 1, +, \times\}$ où l'univers est l'ensemble des nombres complexes $\mathbb{C}$, et qui est muni de deux singletons $0 = \{0\}$ et $1 = \{1\}$ et de deux relations ternaires : $\iota(+) = \{(a, b, c) \mid c = a + b\}$ et $\iota(\times) = \{(a, b, c) \mid c = a \times b\}$. On peut aussi considérer la structure d'univers $\mathbb{N}$ ayant la même signature, avec la même interprétation pour les symboles 0, 1, + et $\times$.

Un graphe $\mathcal{G} = (V, E)$ étiqueté par un alphabet Σ peut aussi être vu comme une structure relationnelle de signature $\{\xrightarrow{a} \mid a \in \Sigma\}$ dont tous les symboles ont une arité 2. L'univers de $\mathcal{G}$ est son ensemble de sommets V et pour tout $a \in \Sigma$, $\iota(\xrightarrow{a}) = \{(x, y) \mid (x, a, y) \in E\}$.

2.5.2 Logique du premier ordre

On définit la logique du premier ordre sur une signature $\mathcal{R}$, qu'on note $\mathsf{FO}(\mathcal{R})$, ou plus simplement, FO, quand il n'y a pas ambiguïté sur $\mathcal{R}$. Dans le cas où on confondra une structure $\mathcal{S}$ avec sa signature, on notera $\mathsf{FO}(\mathcal{S})$ la logique sur la signature de $\mathcal{S}$.

Définition 2.3. *Étant donnés une signature $\mathcal{R}$ et un ensemble dénombrable de variables $\mathcal{V} = \{x, y, z, x_1, x_2, \cdots\}$, une* formule du premier ordre *sur $\mathcal{R}$ est définie récursivement comme suit :*

- **true**
- $R(x_1, \cdots, x_{|R|})$, *avec* $x_1, \cdots, x_{|R|} \in \mathcal{V}$.
- $\phi_1 \wedge \phi_2$ *si* ϕ_1 *et* ϕ_2 *sont des formules du premier ordre.*
- $\neg\phi_1$ *si* ϕ_1 *est une formule du premier ordre.*
- $\exists x, \phi_1$ *si* ϕ_1 *est une formule du premier ordre et* x *est une variable.*

Les deux premiers éléments de la disjonction de cas précédente sont appelés des *atomes*, c'est-à-dire des formules qui ne peuvent pas être décomposées. $\wedge$ et $\neg$ sont appelés des *connecteurs*. $\exists$ est appelé un *quantificateur*.

Une variable x est dite *liée* dans une formule ϕ si elle n'apparaît que dans des formules de la forme $\exists x, \phi_1$. Dans le cas contraire, elle est dite *libre*. Une formule qui ne contient aucune variable libre est dite *close*.

Définition 2.4. *Étant données une signature* $\mathcal{R}$*, une formule* ϕ *de* $\mathsf{FO}(\mathcal{R})$*, une structure* $\mathcal{S} = (\mathcal{U}, \iota)$ *de signature* $\mathcal{R}$ *et une fonction* v *de* $\mathcal{V}$ *dans* $\mathcal{U}$ *appelée* valuation*, on dit que la structure* $\mathcal{S}$ *avec la valuation* v satisfait *la formule* ϕ*, noté* $(\mathcal{S}, v) \models \phi$ *dans les cas suivant :*

- $(\mathcal{S}, v) \models$ **true** *pour toute interprétation* ι *et toute valuation* v.
- $(\mathcal{S}, v) \models R(x_1, \cdots, x_{|R|})$ *si on a* $\iota(R)(v(x_1), \cdots, v(x_{|R|}))$.
- $(\mathcal{S}, v) \models \phi_1 \wedge \phi_2$ *si* $(\mathcal{S}, v) \models \phi_1$ *et* $(\mathcal{S}, v) \models \phi_2$.
- $(\mathcal{S}, v) \models \neg\phi_1$ *si* $(\mathcal{S}, v) \not\models \phi_1$.
- $(\mathcal{S}, v) \models \exists x, \phi_1$ *si il existe un élément* γ *de* Γ *tel que* $(\mathcal{S}, v') \models \phi_1$ *où* $v'(x) = \gamma$ *et* $v'(y) = v(y)$ *pour tout* $y \neq x$.

Étant donnée une formule close ϕ, on notera simplement $\mathcal{S} \models \phi$ si pour toute valuation v, on a $(\mathcal{S}, v) \models \phi$ (la valuation considérée n'ayant pas d'influence, puisque la valeur de toutes les variables apparaissant dans ϕ sera redéfinie par les occurrences de $\exists x$ qui apparaissent forcément avant les occurrences de x). Dans la suite, on utilisera uniquement des formules closes.

On introduit les raccourcis de notation habituels, dont nous nous servirons dans la suite :

- **false** $= \neg$**true**
- $\phi_1 \vee \phi_2 = \neg((\neg\phi_1) \wedge (\neg\phi_2))$
- $\phi_1 \Rightarrow \phi_2 = (\neg\phi_1) \vee \phi_2$
- $\phi_1 \Leftrightarrow \phi_2 = (\phi_1 \Rightarrow \phi_2) \wedge (\phi_1 \Rightarrow \phi_2)$
- $\forall x, \phi_1 = \neg\exists x, (\neg\phi_1)$

Étant donnée une formule close ϕ de $\mathsf{FO}(\mathcal{R})$, l'ensemble des structures $\mathcal{S}$ de signature $\mathcal{R}$ telles que $\mathcal{S} \models \phi$ est appelé le *langage* de ϕ, noté $L(\phi)$. Étant donnée une structure $\mathcal{S}$ de signature $\mathcal{R}$, l'ensemble des formules ϕ telles que $\mathcal{S} \models \phi$ est appelé la théorie de $\mathcal{S}$, notée $\mathrm{Th}(\mathcal{S})$. Étant donnés une signature $\mathcal{R}$ et un symbole relationnel R qui n'est pas dans $\mathcal{R}$, on définit la logique du premier ordre avec R sur $\mathcal{R}$, notée $\mathsf{FO}[R](\mathcal{R})$ comme étant la logique $\mathsf{FO}(\mathcal{R} \cup \{R\})$. Dans la suite, on considérera essentiellement la logique $\mathsf{FO}[\overset{*}{\rightarrow}](\mathcal{G})$ sur un graphe $\mathcal{G}$, où $\overset{*}{\rightarrow}$ est le prédicat d'accessibilité. Cela reviendra à considérer une formule du premier ordre sur le graphe $\mathcal{G}'$ obtenu à partir de $\mathcal{G}$ en ajoutant les arcs (x, τ, y) si il existe un chemin de x à y dans $\mathcal{G}$ et τ n'est pas une étiquette de $\mathcal{G}$.

Exemple 2.2. *On considère la formule* $\phi = \forall x, \exists y, z, +(x, y, z) \wedge 1(z)$ *exprimant que tout nombre possède un inverse par l'addition. Cette formule est satisfaite sur* $\mathbb{C}$*, mais pas sur* $\mathbb{N}$.

2.5.3 Logique du second ordre monadique

On définit maintenant la logique du second ordre monadique sur une signature $\mathcal{R}$, notée $\mathsf{MSO}(\mathcal{R})$. Elle est obtenue en ajoutant un ensemble de variables du second ordre $\mathcal{V}_2 = \{X, Y, Z, X_1, X_2, \cdots\}$ à la définition de la logique du premier ordre, et en ajoutant les cas suivants à la définition d'une formule :

- $x \in X$, pour $x \in \mathcal{V}$ et $X \in \mathcal{V}_2$.
- $\exists X, \phi_1$, où ϕ_1 est une formule de $\mathsf{MSO}(\mathcal{R})$ et $X \in \mathcal{V}_2$.

On étend la définition d'une valuation pour envoyer les variables de $\mathcal{V}_2$ sur des sous-ensembles de $\mathcal{U}$.

On définit la satisfaction d'une formule de $\mathsf{MSO}(\mathcal{R})$ sur une structure $\mathcal{S}$ de signature $\mathcal{R}$ de la même manière que pour une formule de $\mathsf{FO}(\mathcal{R})$ pour les cas communs aux deux, et pour les deux nouveaux cas, on a :

- $(\mathcal{S}, v) \models x \in X$ si $v(x)$ est un élément de $v(X)$.
- $(\mathcal{S}, v) \models \exists X, \phi_1$ si $(\mathcal{S}, v') \models \phi_1$.

Les notions de variables libres et liées s'étendent aux éléments de $\mathcal{V}_2$.

On peut également définir la logique $\mathsf{MSO}[P](\mathcal{R})$ de la même manière qu'on a défini $\mathsf{FO}[P](\mathcal{R})$. Cependant, il est inutile de définir $\mathsf{MSO}[\overset{*}{\to}](\mathcal{G})$ sur un graphe $\mathcal{G}$. En effet, le prédicat $x \overset{*}{\to} y$ peut être défini dans la logique MSO par la formule $\forall X, (x \in X \wedge \forall z, z', (z \in X \wedge z \to z') \Rightarrow z' \in X) \Rightarrow y \in X$. Il est également possible d'exprimer en MSO l'existence d'un chemin étiqueté par un mot appartenant à un langage reconnaissable L, grâce au théorème de Büchi disant qu'on peut exprimer l'appartenance d'un mot à un langage régulier en MSO. La construction de cette formule repose sur la définition de variables du second ordre représentant les états de l'automate et contenant les sommets du graphe de manière à ce que si on arrive dans un sommet x après avoir lu un mot v, x soit étiqueté par les états dans lequel peut être l'automate après avoir lu v. Il faut ensuite imposer que le sommet de départ est étiqueté par un état initial et le sommet d'arrivée l'est par un état final.

On définit enfin la logique du second ordre monadique faible $\mathsf{WMSO}(\mathcal{R})$ de la même manière que $\mathsf{MSO}(\mathcal{R})$, à la différence près que les valuations v ne peuvent envoyer les variables de $\mathcal{V}_2$ que sur des sous-ensembles finis de $\mathcal{U}$.

Chapitre 3

Graphes infinis

Nous avons défini un graphe comme étant la donnée d'un ensemble – potentiellement infini – V de sommets et un ensemble E de relations binaires sur V étiquetées par un alphabet Σ. De ce fait, les formalismes utilisant des relations binaires peuvent s'exprimer comme des graphes, et inversement, ils peuvent être utilisés pour définir des graphes. Ce chapitre vise à faire un rapide état de l'art de plusieurs d'entre eux qui seront utiles dans notre étude.

Le paragraphe 3.1 rappelle dans cette optique les notions d'isomorphisme de graphes, de représentation interne d'un graphe et de transformations de graphes, et liste également les problèmes généraux qui nous intéresseront sur les classes de graphes que nous étudierons. Le paragraphe 3.2 résume les connaissances sur les classes de graphes liées aux automates à pile d'ordre supérieur, et présente le formalisme légèrement différent des systèmes de réécriture de piles qui engendre les mêmes classes de graphes. Le paragraphe 3.3 s'intéresse quant à lui aux classes liées à des systèmes de réécriture suffixe d'arbres et rappelle quelques résultats sur la hiérarchie de graphes dite arbre-automatique.

3.1 Présentations finies de graphes infinis

3.1.1 Isomorphismes de graphes

Un point central de notre travail est la notion d'isomorphisme de graphes. En effet, elle nous permettra de nous concentrer sur la structure des graphes que nous souhaitons étudier et les relations entre les sommets en ignorant l'ensemble exact des sommets.

Définition 3.1. *Étant donnés deux graphes $\mathcal{G}_1 = (V_1, E_1)$ et $\mathcal{G}_2 = (V_2, E_2)$ étiquetés par un même alphabet Σ, on dit qu'ils sont* isomorphes *s'il existe une bijection f de V_1 dans V_2 telle que pour tout $a \in \Sigma$ et tous $x, y \in V_1$, on a $(x, a, y) \in E_1$ si et seulement si $(f(x), a, f(y)) \in E_2$.*

Ainsi, informellement, deux graphes sont isomorphes si en ignorant le nommage exact des sommets, ils sont indistinguables. Dans la suite de ce travail, toutes les classes de graphes que nous définirons le seront à isomorphisme près, et quand nous parlerons d'égalité entre deux classes de graphes, cela sera à isomorphisme près. La plupart du temps, nous laisserons implicite ces isomorphismes pour ne pas surcharger les démonstrations – et parce que les décrire en détail n'apporterait rien à la compréhension.

3.1.2 Questions classiques

Quand on étudie une classe de graphes, il y a plusieurs questions qu'on peut se poser sur les éléments de cette classe. Dans cette thèse, nous étudierons plusieurs classes de graphes, et nous poserons sur ces classes les questions suivantes :

Représentations d'un graphe. La première question reviendra simplement à se demander quelles sont les différentes manières de décrire une classe de graphes. Le sous-paragraphe suivant sera consacré à définir plusieurs manières de décrire ces classes.

Accessibilité. Un autre problème est celui de l'accessibilité d'un sommet y dans un graphe à partir d'un sommet x. Formellement, le problème d'accessibilité dans un graphe $\mathcal{G}$ est le suivant :
Entrée : Deux sommets $x, y \in V_{\mathcal{G}}$.
Sortie : Si $x \xrightarrow[\mathcal{G}]{*} y$.

Un problème naturellement lié est le calcul de l'ensemble des sommets accessibles à partir d'un sommet x. Dans le chapitre 5, nous verrons que dans le cas des systèmes d'addition de vecteurs, le problème de l'accessibilité est décidable, mais que l'ensemble des sommets accessibles à partir d'un sommet donné n'est pas calculable. Pour les classes de graphes dont nous parlerons dans ce chapitre et le suivant, les ensembles d'accessibilité seront des ensembles reconnaissables (dans un sens que nous définirons), et ces ensembles admettront une représentation normalisée, et seront de ce fait calculables.

Vérification de modèle. Le problème de vérification de modèle (model-checking en anglais) consiste à déterminer si une formule logique est vraie sur un graphe ou non (littéralement, déterminer si le graphe est un modèle de la formule considérée). Formellement, le problème de vérification de modèle pour la logique L sur un graphe $\mathcal{G}$ est le suivant :
Entrée : Une formule ϕ dans $L(\mathcal{G})$.
Sortie : Si $\mathcal{G} \models \phi$.

Ce problème revient à être capable de décider l'ensemble $\mathrm{Th}(\mathcal{G})$.

Plus précisément, on cherchera à déterminer des classes de graphes $\mathcal{C}$ telles que pour tout $\mathcal{G} \in \mathcal{C}$, le problème de vérification de modèle pour L sur $\mathcal{G}$ est décidable. Dans la suite, on dira que $\mathcal{G}$ a une L-théorie décidable si le problème de vérification de modèle pour la logique L est décidable sur $\mathcal{G}$.

Langage reconnu. Les graphes sont souvent utilisés comme accepteurs de langages. Nous nous intéresserons aussi à cet aspect en nous demandant quel langage est reconnu par un graphe. Pour cela, on a besoin de savoir où commencer la lecture d'un mot, et - éventuellement - où la terminer. On a donc deux types de langages :

Le langage des traces d'un graphe $\mathcal{G}$ à partir d'un ensemble de configurations initiales I est l'ensemble $\mathcal{L}(\mathcal{G}, I) = \{u \mid \exists q \in I, q' \in V, q \xrightarrow{u} q'\}$.

Le langage d'un graphe $\mathcal{G}$ entre l'ensemble de configurations initiales I et l'ensemble de configurations finales F est l'ensemble $L(\mathcal{G}, I, F) = \{u \mid \exists q \in I, q' \in F, q \xrightarrow{u} q'\}$.

Étudier les langages reconnus par un graphe entre deux ensembles quelconques n'a pas d'intérêt, puisque pour déterminer ces langages, il est nécessaire d'être capable de déterminer si un sommet du graphe est initial ou final. Aussi, dans la suite, les langages que nous étudierons seront définis entre deux ensembles de sommets finis, ou plus généralement entre deux ensembles de sommets définissables dans une logique L décidable sur le graphe considéré, ce qui permettra de calculer effectivement ces langages.

Pour les classes de graphes considérées dans ce chapitre et le suivant, la question sera de savoir où ces langages se situent dans la hiérarchie de Chomsky – et si l'on peut en trouver une représentation non directement liée à cette classe de graphes. Dans le chapitre 5, la question sera de déterminer dans quels cas le langage reconnu par un graphe est algébrique.

Propriétés de clôture. Comme on le verra dans le sous-paragraphe 3.1.4, il existe différentes transformations possibles applicables sur un graphe. Une question est de déterminer, étant donnée une classe de graphes, par quelles transformations elle est close. Formellement, une classe de graphes $\mathcal{C}$ est close par une transformation θ si pour tout $\mathcal{G} \in \mathcal{C}$, $\theta(\mathcal{G}) \in \mathcal{C}$.

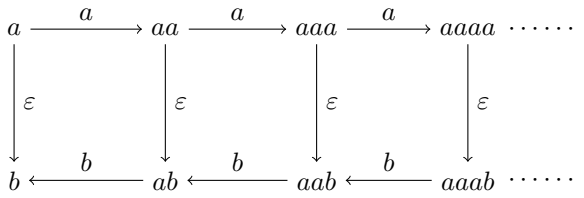


FIGURE 3.1 – Un graphe infini : l'échelle

3.1.3 Représentation interne

On rappelle qu'un graphe étiqueté par Σ est la donnée d'un ensemble de sommets V et d'un ensemble d'arcs $E \subseteq V \times \Sigma \times V$. Une représentation interne est la donnée explicite des arcs comme des relations sur V. Ainsi, une représentation interne d'un graphe (V, E) est la donnée de $\mathcal{R} = \{R_a \mid a \in \Sigma\}$ telle que pour tous éléments x, y de V, on a $(x, a, y) \in E$ si et seulement si $(x, y) \in R_a$. La représentation interne est dite finie si tous les R_a peuvent être décrites finiment. Dans la suite, on s'intéressera uniquement à des représentations internes finies.

Exemple 3.1. *On considère le graphe représenté à la figure 3.1. Sa représentation interne est la suivante :*

- $V = a^* \cdot \{a, b\}$
- $E = \{(u.a, a, u.a.a), (u.a, \varepsilon, u.b), (u.a.b, b, u.b) \mid u \in \{a, b\}^*\}$.

On verra par la suite que ce graphe est la restriction par accessibilité à partir du sommet a du graphe d'un système de réécriture de mots.

Les représentations internes permettent de visualiser la structure du graphe assez aisément, et surtout de décrire ces graphes de manière automatique, la description finie des relations en étant la clé. Ces représentations finies dériveront notamment de systèmes de réécriture de mots ou d'arbres, ou de formalismes de calculs comme les automates ou les systèmes d'addition de vecteurs, par le biais de leur relation de transition. En effet, ces machines procurent une représentation explicite de leur relation de transition. Les paragraphes suivants de ce chapitre présenteront des graphes dont les représentations internes sont liées à des systèmes de réécriture de mots ou d'arbres et à des automates à pile d'ordre supérieur.

3.1.4 Transformations de graphes

On va maintenant décrire des transformations permettant de décrire des graphes en fonction d'autres graphes. Une telle description est appelée «représentation externe», par opposition à la représentation interne, dont elle diffère par une réelle anonymie des sommets. La plupart de ces transformations induiront la décidabilité de certaines logiques sur le graphe d'arrivée en fonction des propriétés décidables sur le graphe de départ. L'intérêt sera ensuite de trouver des équivalences entre des représentations internes et externes de classes de graphes dans le but d'obtenir le plus de propriétés possibles sur ces classes.

L-interprétation. Une L-interprétation (où L est une logique, par exemple, MSO, FO, etc...) est un uplet $\mathcal{I} = (\delta(x), (\phi_a(x, y))_{a \in \Sigma})$ de formules de la logique L exprimées sur la signature $\mathcal{S}$ d'un graphe. Si on l'applique à un graphe $\mathcal{G} = (V, E)$ de signature $\mathcal{S}$, on obtient le graphe $\mathcal{I}(\mathcal{G}) = (V', E')$ étiqueté par l'alphabet Σ tel que :

- $V' = \{x \in V \mid \mathcal{G} \models \delta(x)\}$
- $E' = \{(x, a, y) \mid \mathcal{G} \models \phi_a(x, y)\}$.

On dérive facilement de cette définition les propriétés suivantes :

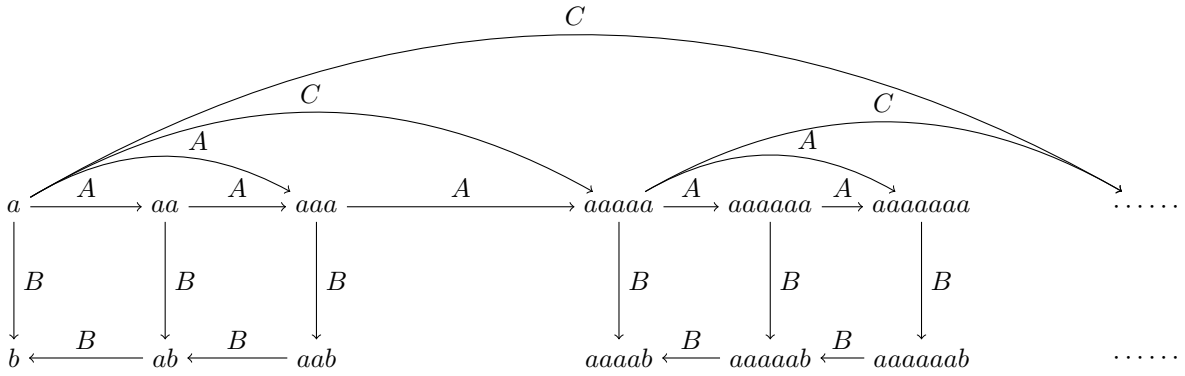


FIGURE 3.2 – Un graphe obtenu par MSO-interprétation de l'échelle

Proposition 3.1. *La composition de deux L-interprétations est une L-interprétation.*

Proposition 3.2. *Une L-interprétation préserve la décidabilité de la L-théorie. C'est-à-dire que si $\mathcal{G}$ possède une L-théorie décidable, alors $\mathcal{I}(\mathcal{G})$ possède une L-théorie décidable.*

Supposons que l'on dispose d'une L-interprétation d'un graphe $\mathcal{G}$ dans un graphe $\mathcal{G}'$. Ces deux propositions s'obtiennent en montrant que pour toute L-formule ϕ sur $\mathcal{G}'$, on peut construire une L-formule ψ sur $\mathcal{G}$ qui lui est équivalente, c'est-à-dire que $\mathcal{G}' \models \phi$ si et seulement si $\mathcal{G} \models \psi$. Pour ce faire, on définit ψ comme étant ϕ dans laquelle on a remplacé les formules atomiques $x \xrightarrow{a} y$ par les formules de l'interprétation $\phi_a(x, y)$. Comme les ϕ_a sont des L-formules, ψ est bien une L-formule. De plus, par définition de l'interprétation, on a $\mathcal{G}' \models x \xrightarrow{a} y$ si et seulement si $\mathcal{G} \models \phi_a(x, y)$. On en déduit bien que $\mathcal{G} \models \psi$ si et seulement si $\mathcal{G}' \models \phi$. Ainsi, si $\mathcal{G}$ possède une L-théorie décidable, toute L-formule sur $\mathcal{G}'$ étant équivalente à une L-formule sur $\mathcal{G}$ est donc décidable. De même, toute L-interprétation sur $\mathcal{G}'$ peut être traduite en une L-interprétation sur $\mathcal{G}$.

Exemple 3.2. *Si on reprend l'exemple 3.1 de l'échelle, on peut définir sur ce graphe la MSO-interprétation $\mathcal{I} = (\delta(x), \phi_A(x, y), \phi_B(x, y), \phi_C(x, y))$, avec :*

- $\delta(x) = \exists y \forall X((\forall w(\forall w', \neg w' \to w) \Rightarrow w \in X) \wedge \forall z, z', z \in X \wedge z \xrightarrow{a^4} z' \Rightarrow z' \in X) \Rightarrow y \in X \wedge (x = y \vee y \xrightarrow{a} x \vee y \xrightarrow{aa} x \vee y \xrightarrow{\varepsilon} x \vee y \xrightarrow{a\varepsilon} x \vee y \xrightarrow{aa\varepsilon} x)$.
- $\phi_A(x, y) = x \xrightarrow{a} y \vee x \xrightarrow{aa} y$.
- $\phi_B(x, y) = x \xrightarrow{b} y \vee x \xrightarrow{\varepsilon} y$.
- $\phi_C(x, y) = \forall X, (a \in X \wedge \forall z, z', z \in X \wedge z \xrightarrow{a^4} z' \Rightarrow z' \in X) \Rightarrow x \in X \wedge x \xrightarrow{aaa} y$.

Le graphe obtenu en appliquant $\mathcal{I}$ au graphe de l'exemple 3.1 est représenté à la figure 3.2.

Interprétation à ensembles finis. Une transformation proche de la L-interprétation est l'interprétation à ensembles finis. Contrairement à la précédente, elle ne définit pas un graphe sur un sous-ensemble des sommets du graphe d'origine, mais sur les sous-ensembles finis des sommets du graphe d'origine.

Une interprétation à ensembles finis (ou FSI) est un uplet $\mathcal{I} = (\delta(X), (\phi_a(X, Y))_{a \in \Sigma})$ de WMSO-formules exprimées sur la signature $\mathcal{S}$ d'un graphe. Si on l'applique à un graphe $\mathcal{G} = (V, E)$ de signature $\mathcal{S}$, on obtient le graphe $\mathcal{I}(\mathcal{G}) = (V', E')$ étiqueté par l'alphabet Σ tel que :

- $V' = \{X \in \mathcal{P}_f(V) \mid \mathcal{G} \models \delta(X)\}$
- $E' = \{(X, a, Y) \mid \mathcal{G} \models \phi_a(X, Y)\}$.

Par définition, on a la propriété suivante :

Proposition 3.3. *Si $\mathcal{G}$ possède une* WMSO*-théorie décidable, alors $\mathcal{I}(\mathcal{G})$ possède une FO-théorie décidable.*

Comme dans le cas des L-interprétations, cette propriété repose sur l'observation qu'une FO-formule sur $\mathcal{I}(\mathcal{G})$ peut être transformée en une MSO-formule équivalente sur $\mathcal{G}$ en remplaçant les formules atomiques $x \xrightarrow{a} y$ par les MSO-formules $\phi_a(X, Y)$ (et en remplaçant les variables du premier ordre par des variables du second ordre).

Cette transformation a été introduite et étudiée par Colcombet et Löding dans [CL07]. Elle jouera un rôle centrale dans la classe de graphes que nous définirons au chapitre 4.

Restriction à un ensemble de sommets. La restriction d'un graphe à un ensemble de sommets permet de ne regarder qu'une partie du graphe sans modification, en ayant simplement éliminé des sommets - et les arcs entre ces sommets et ceux conservés. La restriction d'un graphe $\mathcal{G} = (V, E)$ étiqueté par Σ à un ensemble de sommet V', est le graphe $\mathcal{G}_{|V'} = (V', E \cap V' \times \Sigma \times V')$.

A priori, la restriction à un ensemble quelconque de sommets ne conserve aucune propriété de décidabilité logique. Cependant, si l'ensemble V' est définissable dans une logique L, alors la restriction à L est simplement une L-interprétation dont la formule $\delta(x)$ est la définition de L et les $\phi_a(x, y)$ sont simplement les formules atomiques $x \xrightarrow{a} y$. Et dans ce cas la décidabilité de L est préservée par la restriction à L. On utilisera la plupart du temps la restriction dans ce cadre, et cela nous permettra de gagner en concision en parlant simplement de restriction plutôt que d'interprétation (qui est une transformation très expressive, et de ce fait peu aisée à manipuler).

Substitution inverse. Étant donné un graphe $\mathcal{G} = (V, E)$ étiqueté par Σ et un morphisme h de Σ dans $\mathcal{P}(\Sigma^*)$, l'image de $\mathcal{G}$ par l'inverse de h est le graphe $h^{-1}(\mathcal{G}) = (V, \{(u, a, v) \mid \exists w \in h(a), u \xrightarrow{w} v\})$.

Cette transformation a été introduite par Caucal dans [Cau96], avec le cas des substitutions rationnelles inverses, c'est-à-dire avec un morphisme h tel que pour tout a, $h(a)$ est un langage rationnel. Il s'agit d'un cas particulier d'interprétation monadique, puisque tout langage rationnel peut être défini par une MSO-formule. De ce fait, cette transformation préserve également la décidabilité de MSO. Elle permet de donner une vision plus minimale de l'expressivité nécessaire à la réalisation d'une transformation donnée que les L-interprétations. Par exemple, la classe des graphes suffixe-reconnaissables peut-être définie comme l'ensemble des MSO-interprétations de l'arbre binaire complet ou comme l'ensemble des substitutions inverses appliquées à l'arbre binaire complet.

Structure arborescente. La structure arborescente d'un graphe $\mathcal{G} = (V, E)$ étiqueté par Σ peut être vue comme une «itération» du graphe, c'est-à-dire que chaque sommet de la structure arborescente représente un mot de sommets du graphe d'origine. Pour tout mot u de V^*, la restriction de la structure arborescente de $\mathcal{G}$ aux sommets de la forme $u.x$, avec $x \in V$ est isomorphe à $\mathcal{G}$, et on ajoute des arcs de «copie», permettant de passer des sommets de la forme $u.x$ aux sommets de la forme $u.x.x$. Formellement, sa structure arborescente est le graphe

$$\text{Treegraph}_{\#}(\mathcal{G}) = (V^+, \{(w.u, a, w.v) \mid (u, a, v) \in E\} \cup \{(w.u, \#, w.u.u)\})$$

où $\#$ est un symbole n'appartenant pas à Σ.

La première notion de structure arborescente a été introduite par Shelah [She75]. La structure arborescente considérée par Shelah est la suivante :

$$(V^+, \{(w.u, a, w.v) \mid (u, a, v) \in E\} \cup \{(w, \text{son}, w.u \mid w \in V^*, u \in V\})$$

Dans [Tho90], Thomas attribue à Stupp la preuve que cette forme préserve la décidabilité de MSO. Dans [Sem84], Semenov attribue à Muchnick la notion de structure arborescente suivante :

$$(V^+, \{(w.u, a, w.v) \mid (u, a, v) \in E\} \cup \{(w, \text{son}, w.u \mid w \in V^*, u \in V\} \cup \{(w.u, \#, w.u.u)\})$$

et il donne un schéma de preuve que cette structure préserve la décidabilité de MSO. La première preuve complète de ce résultat est due à Walukiewicz dans [Wal96, Wal02].

Exemple 3.3. *On considère à nouveau l'échelle de l'exemple 3.1, et on représente en partie sa structure arborescente à la figure 3.3.*

Dépliage. Une opération plus simple que la structure arborescente est le dépliage d'un graphe à partir d'un sommet donné r. Il peut être vu comme l'arbre des chemins du graphe ayant r comme premier sommet. Le dépliage d'un graphe $\mathcal{G} = (V, E)$ étiqueté par Σ à partir du sommet $r \in V$ est un arbre $\mathrm{Unf}(G, r) = (r(\Sigma V)^*, E')$ avec $E' = \{(w.u, a, w.u.a.v) \mid (u, a, v) \in E\}$. Courcelle et Walukiewicz [CW98] ont montré que le dépliage d'un graphe préserve également la décidabilité de MSO. Ce résultat peut s'obtenir en définissant le dépliage d'un graphe par une MSO-interprétation de sa structure arborescente [Car06]. Colcombet a ensuite montré que la structure arborescente d'un arbre déterministe t peut être obtenue par $h_2^{-1}(\mathrm{Unf}(h_1^{-1}(t), r))$, où h_1 et h_2 sont des substitutions rationnelles et r la racine de t [Col04].

Exemple 3.4. *On présente à la figure 3.4 le dépliage de l'échelle à partir du sommet a.*

Générateurs. Une notion liée à la définition des classes de graphes est celle de *générateur*. On dit qu'un graphe $\mathcal{G}$ est *générateur* d'une classe de graphes $\mathcal{C}$ pour un type de transformation si pour tout $\mathcal{G}' \in \mathcal{C}$, il existe une transformation t de ce type tel que $\mathcal{G}' = t(\mathcal{G})$. On verra par exemple par la suite que l'ensemble des graphes associés à des automates à pile est engendré par la structure arborescente d'un graphe fini via des MSO-interprétations. Il est également possible d'obtenir des classes de graphes via un générateur et une famille de transformations.

Systèmes d'équations. Une dernière description de graphes dont nous parlerons peu est la représentation équationnelle. On considère un ensemble $\mathcal{O}$ d'opérations sur des graphes, et on définit un système d'équation comme étant un terme t (potentiellement infini) composé d'opérations de $\mathcal{O}$. La solution d'un tel terme est le plus petit point fixe satisfaisant le terme t. Il est également possible de considérer des graphes étiquetés par des opérations de $\mathcal{O}$. On peut alors les utiliser pour définir des systèmes d'équations en les dépliant à partir d'un de leur sommet (on obtient alors un terme infini).

La notion de systèmes d'équation comme description de structures algébriques est ancien et peut être retracé au moins à [MW67]. Pour ce qui concerne les graphes, l'une des première représentation équationnelle de graphes a été défini par Courcelle [Cou89] avec les graphes HR-équationnels, qui est une représentation liée à celle des grammaires de graphes. Un autre système d'opérations de graphes a été introduit par Barthelman [Bar97] : les opérateurs VR. Ces opérateurs manipulent des graphes dont les sommets sont colorés et contiennent une opération définissant un graphe contenant un unique sommet d'une couleur donnée, une opération d'union, une opération d'ajout d'arcs entre les sommets de deux couleurs et une opération de recoloriage des sommets. Colcombet a étendu ces opérateurs aux opérateurs VRS [Col04] et VRA [Col02] en ajoutant respectivement un opérateur de produit synchronisé et un opérateur de produit asynchrone.

Cette représentation n'est pas au centre de notre étude, nous nous bornerons donc à citer les résultats qui en font mention sans donner de détail techniques. Cependant, toutes les classes dont nous parlerons dans ce chapitre disposent d'une représentation équationnelle, avec les opérateurs VR, VRA ou VRS, et une extension possible de notre étude est de se demander si les graphes définis au chapitre 4 disposent d'une représentation équationnelle, dont nous conjecturons qu'elle fait intervenir les opérateurs VRA.

Exemple 3.5. *Comme il est compliqué de donner ici la définition formelle et complète de ces systèmes, nous donnons un exemple très simple d'un graphe défini par une équation sur les*

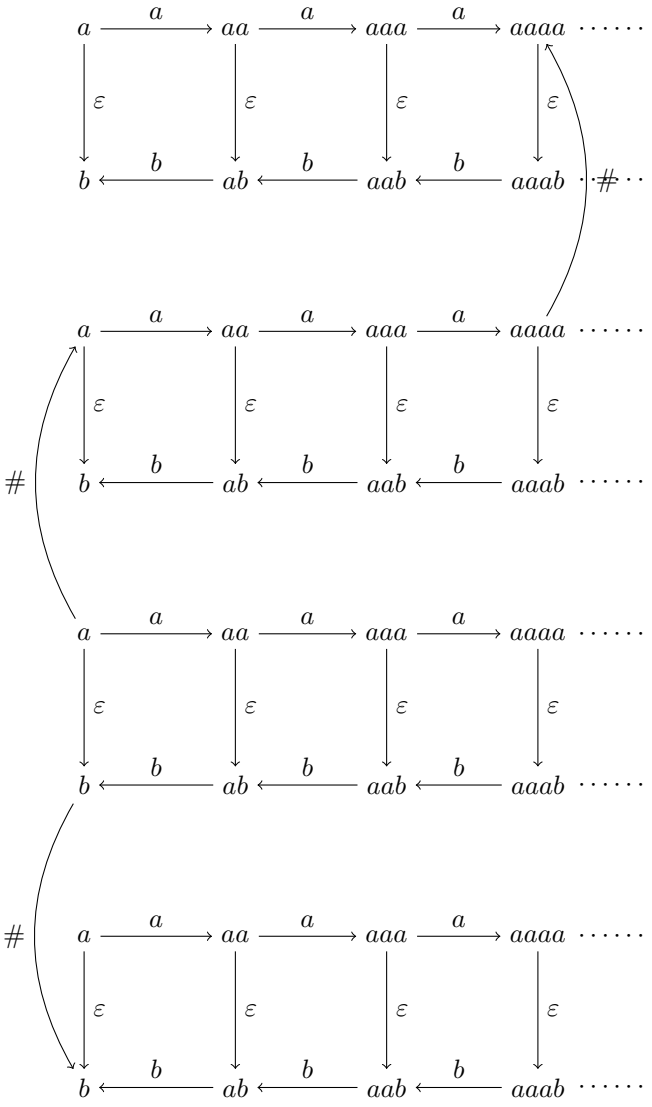


FIGURE 3.3 – Une partier de la structure arborescente du graphe de l'exemple 3.1.

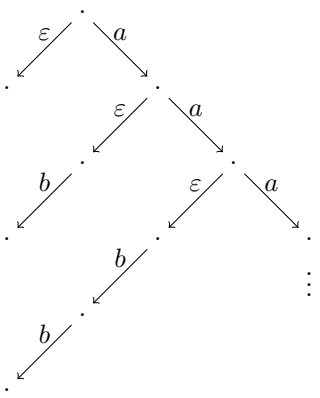


FIGURE 3.4 – Le dépliage de l'échelle à partir du sommet a.

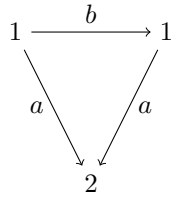


FIGURE 3.5 – Le graphe solution du terme de l'exemple 3.5.

opérateurs VR. On considère le terme

$$t = \mathrm{add}_{1,2,a}(\oplus(2, \mathrm{recol}_c(\mathrm{add}_{1,2,b}(\oplus(1,2)))))$$

où c est la fonction de recoloriage $\{1 \to 1, 2 \to 1\}$. Le graphe défini par ce terme est représenté en figure 3.5. Le terme génère d'abord deux sommet entre lesquels il ajoute un arc étiqueté par b. Ensuite, il ajoute entre chacun de ces deux sommets et un nouveau sommet, un arc étiqueté par a.

3.2 Graphes liés aux piles

On présente ici les classes de graphes associés aux automates à pile et aux automates à pile d'ordre supérieur. On commencera par rappeler une définition classique des automates à piles, et les graphes que l'on peut définir à partir de ces automates. Ensuite nous résumerons les propriétés et les différentes descriptions connues de ces classes de graphes, et on donnera un bref historique de leur introduction. Puis, nous effectuerons ce travail sur les automates à pile d'ordre supérieur. Enfin, nous donnerons un formalisme équivalent aux automates à piles d'ordre supérieur (dans le sens où ils définiront les mêmes classes de graphes), que nous intitulerons systèmes de réécriture de piles et qui nous sera utile dans la définition des systèmes de réécriture d'arbres de piles introduits au chapitre 4.

3.2.1 Automates à pile

Les automates à pile sont un modèle classique en théorie des langages qui reconnaissent les langages algébriques. Il existe de nombreuses définitions équivalentes d'automates à pile. Nous présentons ici une définition dite faible des automates à piles, dans le sens où nous imposons

que l'automate n'effectue qu'une action simple sur la pile par transition, et normalisée dans le sens où nous imposons que, depuis un état donné, l'automate puisse uniquement emprunter des transitions étiquetées par une lettre ou uniquement des transition étiquetées par ε.

Un automate à pile est défini comme un uplet $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, F, \Delta)$ avec :

- Q un ensemble fini d'états, découpé entre les états observables Q_{Obs} et les états internes Q_{Int}.
- Σ un alphabet d'entrée fini,
- Γ un alphabet de piles fini,
- $q_0 \in Q_{\text{Obs}}$ l'état *initial*,
- $F \subseteq Q_{\text{Obs}}$ l'ensemble des états *finaux*,
- $\Delta \subseteq Q_{\text{Obs}} \times (\{\text{push}_\alpha, \text{pop}_\alpha \mid \alpha \in \Gamma\} \cup \{\varepsilon\}) \times \Sigma \times Q \cup Q_{\text{Int}} \times (\{\text{push}_\alpha, \text{pop}_\alpha \mid \alpha \in \Gamma\} \cup \{\varepsilon\}) \times \{\varepsilon\} \times Q$.

Une configuration de $\mathcal{A}$ est un couple (q, w) avec q un état de Q et $w \in \Gamma^*$ une pile. La configuration initiale de $\mathcal{A}$ est (q, ε), et l'ensemble des configurations finales est $F \times \Gamma^*$. L'automate à pile induit les relations $\xrightarrow[\mathcal{A}]{a}$ pour tout $a \in \Sigma \cup \{\varepsilon\}$ définies par : $(q, u) \xrightarrow[\mathcal{A}]{a} (q', v)$ si il existe une transition de Δ, $\tau = (q, \text{push}_\alpha, a, q')$ et $v = u\alpha$, ou $\tau = (q, \text{pop}_\alpha, a, q')$ et $u = v\alpha$, ou $\tau = (q, \varepsilon, a, q')$ et $u = v$.

Une propriété jouant un rôle central dans notre étude est que pour tout automate à pile, l'ensemble des piles pouvant apparaître dans une configuration finale de l'automate à pile est un ensemble reconnaissable de mots. L'une de nos préoccupations sera d'étendre ces propriétés aux modèles que nous définirons par la suite.

Proposition 3.4. *L'ensemble* $\{w \mid \exists q \in F, (q_0, \varepsilon) \xrightarrow[\mathcal{A}]{*} (q, w)\}$ *est un ensemble reconnaissable de mots.*

Les les automates à pile reconnaissent les langages algébriques qui forment le niveau 2 de la hiérarchie de Chomsky, c'est-à-dire sont engendrés par des grammaires algébriques (aussi connues sous le nom de grammaires hors-contextes). Cette classe de langages est close par union, mais ni par complémentaire ni par intersection. Elle est également strictement incluse dans les langages contextuels et contient strictement les langages rationnels.

Le graphe canoniquement associé à un automate à pile est $\mathcal{G}_\mathcal{A} = (Q \times \Gamma^*, \{((q, w), a, (q', w')) \mid (q, w) \xrightarrow[\mathcal{A}]{a} (q', w')\})$. Cependant, on considérera dans la suite la restriction de ce graphe à l'ensemble des configurations accessibles depuis la configuration initiale (qu'on notera $\mathcal{G}_{\mathcal{A},(q_0,\varepsilon)}$) ou à un ensemble rationnel de sommets. On considérera également $\mathcal{G}_{\mathcal{A},T}$ le graphe des transitions de l'automate à pile qui est obtenu en ne conservant que les configurations contenant un état observable de l'automate à pile et en contractant les arcs étiqueté par ε. Formellement, on a $\mathcal{G}_{\mathcal{A},T} = (V, E)$ avec

- $V = \{(q, w) \mid q \in Q_{\text{Obs}} \wedge (q_0, \varepsilon) \xrightarrow[\mathcal{A}]{*} (q, w)\}$,
- $E = \{((q_1, w_1), a, (q_2, w_2)) \mid (q_1, w_1) \xrightarrow[\mathcal{A}]{a\varepsilon^*} (q_2, w_2)\}$.

Exemple 3.6. *On considère l'automate à pile* $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{A\}, q_0, \{q_3\}, \Delta)$ *avec*

$$\begin{aligned}\Delta = \{&(q_0, \text{push}_B, a, q_1),\\ &(q_1, \text{push}_A, a, q_1),\\ &(q_1, \text{pop}_A, b, q_2),\\ &(q_2, \text{pop}_A, b, q_2),\\ &(q_2, \text{pop}_B, b, q_3),\\ &(q_1, \text{pop}_B, b, q_3)\}\end{aligned}$$

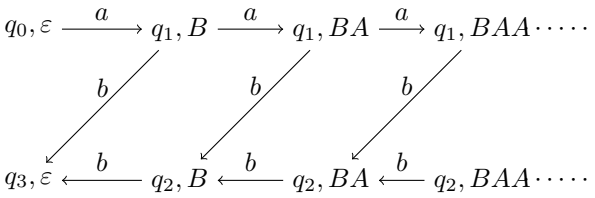


FIGURE 3.6 – Le graphe enraciné en (q_0, ε) d'un automate à pile

Tous les états sont observables.

Le langage reconnu par $\mathcal{A}$ est $L(\mathcal{A}) = \{a^n b^n \mid n \geq 1\}$. En effet, l'automate commence par lire un a en empilant un B et passe dons l'état q_1. Dans cet état, l'automate lit des a en empilant un A à chaque lecture, avant de passer dans l'état q_2 où il lit des b en dépilant des A à chaque lecture. Il ne peut passer dans l'état final q_3 que lorsqu'il peut dépiler un B, empilé à la première transition du calcul, et a ainsi lu autant de a que de b.

La restriction à l'ensemble des sommets accessibles depuis la configuration initiale du graphe associé à cet automate à pile est représenté à la figure 3.6. Comme cet automate à pile ne contient pas de transition étiqueté par ε, son graphe des transitions est identiques au précédent.

Systèmes de réécriture suffixe de mots.

Un formalisme lié à celui des automates à piles est celui des systèmes de réécriture suffixes de mots, dans le sens où les deux formalismes peuvent être utilisés pour engendrer les mêmes graphes. Une règle de réécriture suffixe de mots est un couple de mots (g, d) sur Γ^*. La relation définie par son application suffixe est $\mathrm{r}_{(g,d)}(u, v)$ si et seulement si il existe un mot w tel que $u = wg$ et $v = wd$. Un système de réécriture suffixe de mots étiqueté par Σ est la donnée de $|\Sigma|$ ensembles de règles R_a. Il est dit fini si tous les R_a sont finis. Il est dit reconnaissable si pour tout a, R_a est une union finie d'ensembles de la forme $U \times V$ avec U et V des ensembles reconnaissables de mots. Un système de réécriture suffixe de mots R définit naturellement un graphe $\mathcal{G}_R = (V, E)$ avec $V = \Gamma^*$ et $E = \{(u, a, v) \mid \exists (g, d) \in R_a, \mathrm{r}_{(g,d)}(u, v)\}$.

Dans [Cau96], Caucal introduit une extension des systèmes de réécriture suffixe de mots : les relations suffixe-reconnaissables [1]. Une règle suffixe-reconnaissable est un ensemble de la forme $W(U \times V)$, avec U, V, W des ensembles reconnaissables de mots. La relation qu'elle définit est $\{(u, v) \mid \exists w \in W, l \in U, r \in V, u = wl \wedge v = wr\}$. Autrement dit, en plus de réécrire un suffixe du mot, la relation vérifie que le préfixe conservé appartient à un ensemble reconnaissable. Une règle de réécriture suffixe est donc simplement une règle suffixe-reconnaissable avec $W = \Gamma^*$. Une relation suffixe-reconnaissable est définie par une union finie de règles suffixe-reconnaissables. Caucal montre également que les relation suffixe-reconnaissables forment une algèbre de Boole.

Graphes d'automates à pile

La première mention de graphes liés à des automates à pile est due à Müller et Schupp dans [MS85]. Ils montrent que les graphes d'automates à pile restreints aux sommets accessibles depuis la configuration initiale ont une MSO-théorie décidable en montrant que l'ensemble des sous-graphes connexes obtenus en se restreignant aux sommets non-atteignables en moins de k pas pour tout k, est fini à isomorphisme près et en utilisant le théorème de Rabin.

Théorème 3.5 (Rabin, [Rab69])**.** *L'arbre binaire complet a une* MSO*-théorie décidable.*

1. La dénomination habituelle de ces relations est «préfixe-reconnaissable», Caucal ayant choisi de considérer la réécriture préfixe de mots, qui est équivalente à la réécriture suffixe de mots. Cependant nous étendrons plus tard ces notions aux arbres, pour lesquels les réécritures préfixes et suffixes ne sont pas équivalentes. Comme nous étudierons la réécriture suffixe d'arbres, nous présentons la version suffixe-reconnaissable.

Dans [Cau92], Caucal montre que ces graphes sont isomorphes aux graphes HR-équationnels de degré finis possédant une racine définis par Courcelle dans [Cou90]. Dans [Cau95], il montre que les graphes HR-équationnels sont isomorphes aux graphes d'automates à piles où la restriction par accessibilité est remplacée par une restriction à un ensemble rationnel de configurations (c'est-à-dire que dans la définition précédente, on remplace V par $\{(q, w) \mid q \in Q \wedge w \in L\}$ où L est un langage rationnel), et à la clôture à droite des systèmes finis de réécriture de mots.

Dans [Cau96], Caucal introduit les graphes suffixe-reconnaissables, qui disposent des représentations présentées dans le théorème suivant (chacune étant attribuée à son auteur). Dans ces représentations, on élimine implicitement les sommets isolés par une restriction.

Théorème 3.6. *Étant donné un graphe $\mathcal{G} = (V, E)$, les caractérisations suivantes sont équivalentes, à isomorphisme près :*

1. *$E = \{(u, a, v) \mid (u, v) \in \mathcal{R}_a\}$, où pour tout $a \in \Sigma$, $\mathcal{R}_a$ est une relation suffixe-reconnaissable, et V est l'ensemble des mots de Γ^* qui apparaissent dans l'une des $\mathcal{R}_a$. Ce type de graphe est appelé suffixe-reconnaissable [Cau96].*
2. *$\mathcal{G} = \mathcal{G}'_{|W}$, où W est un ensemble rationnel de mots et $\mathcal{G}'$ est le graphe d'un système reconnaissable de réécriture suffixe de mots [Cau96].*
3. *V est un ensemble rationnel de mots et $E = \{((q, w), a, (q', w')) \mid (q, w) \xrightarrow[\mathcal{A}]{\varepsilon^* a \varepsilon^*} (q', w')\}$. Ces graphes sont appelés graphes de transition de l'automate à pile $\mathcal{A}$ [Sti00]. Notons que l'ensemble des sommets n'est pas nécessairement restreint à l'ensemble des configurations accessibles de l'automate.*
4. *$\mathcal{G}$ est le graphe de type Cayley d'un système de réécriture de mots avec chevauchements préfixes [CK02]. un*
5. *$\mathcal{G}$ peut être construit à partir de l'arbre binaire complet par une* MSO*-interprétation [Blu01].*
6. *$\mathcal{G}$ peut être obtenu par l'application d'une substitution rationnelle inverse au dépliage d'un graphe fini [Cau96].*
7. *$\mathcal{G}$ est VR-équationnel [Bar97].*

Cette variété de représentations fait des graphes suffixes-reconnaissables une classe assez robuste, et la plupart de ses propriétés découlent facilement de certaines présentations. Par exemple, la décidabilité de la MSO-théorie de ces graphes provient facilement de (5) et de (6), grâce au théorème de Rabin sur la décidabilité de la MSO-théorie de l'arbre binaire complet. Cette représentation permet également de déduire que la classe des graphes suffixe-reconnaissables est close par MSO-interprétation (et donc par ses dérivés, restriction rationnelle et substitution rationnelle inverse), et encore que l'arbre binaire complet est un générateur de cette classe par MSO-interprétation. Le fait que les traces de ces graphes sont les langages algébriques provient lui du cas (3).

Récapitulons les propriétés que nous avons présentées sur les graphes suffixe-reconnaissables, en insistant sur les réponses aux problèmes présentés au début de ce chapitre :

- Ils disposent des représentations présentées au théorème précédent.
- Ils possèdent une MSO-théorie décidable.
- L'ensemble des sommets accessibles à partir d'un sommet donné est un ensemble rationnel de mots (dans les présentations internes), et est donc calculable.
- Les langages reconnus par ces graphes sont les langages algébriques.
- Cette classe de graphes est close par union, MSO-interprétation, restriction à des ensembles rationnels de sommets et substitution rationnelle inverse.
- L'arbre binaire complet est un générateur de cette classe par MSO-interprétation et par substitution rationnelle inverse.

3.2.2 Automates à pile d'ordre supérieur

Dans [Gre70], Greibach attribue à Aho et Ullman la paternité des automates à pile d'ordre 2, et la preuve qu'ils reconnaissent les langages indexés (introduits par Aho dans [Aho68]). Maslov [Mas74, Mas76] a ensuite défini les automates à pile d'ordre supérieur à 2 et montré que les langages qu'ils reconnaissent sont les langages définis par les grammaires indexées de niveau équivalent. Ces grammaires sont équivalentes aux grammaires OI de même ordre [Dam82]. On présente le modèle d'automates à piles considéré par Carayol dans [Car05].

Intuitivement, une pile d'ordre 1 est une suite de lettres, une pile d'ordre 2 est une suite non vide de piles d'ordre 1, etc. L'ensemble des piles d'ordre 1, ou 1-piles, sur l'alphabet Γ est $\mathit{Stacks}_1(\Gamma) = \Gamma^*$. Pour $n > 1$, une pile d'ordre n est une suite non-vide de piles d'ordre $n-1$, et l'ensemble des piles d'ordre n est donc $\mathit{Stacks}_n(\Gamma) = \mathit{Stacks}_{n-1}(\Gamma)^+$. La pile d'ordre 1 correspondant au mot $\alpha_1 \cdots \alpha_k$ sera notée $p = [\alpha_1 \cdots \alpha_k]_1$, k sera appelé la *taille* de la pile et noté $|p|$, α_k son *sommet de pile*, et α_1 son *fond de pile*. La pile correspondant à la suite vide ε sera appelée pile vide et notée $[]_1$. De manière analogue, on note $p = [p_1 \cdots p_k]_n$ une pile d'ordre n, ou n-pile, la taille, le sommet et le fond de pile étant définis de la même manière qu'au niveau 1. Il n'y a pas de pile vide d'ordre n, cependant on définit inductivement la pile $[]_n = [[]_{n-1}]_n$, pour simplifier les notations.

On définit formellement un ensemble d'opérations applicables aux piles, respectant la politique d'accès de pile, c'est-à-dire s'appliquant toujours au sommet. Au niveau 1, on conserve simplement les opérations apparaissant dans les automates à piles : $\mathrm{Ops}_1(\Gamma) = \{\mathrm{push}_\alpha, \mathrm{pop}_\alpha \mid \alpha \in \Gamma\} \cup \{\varepsilon\}$. Ces opérations définissent les relations suivantes :

- $\mathrm{r}_{\mathrm{push}_\alpha}(p, p')$ si et seulement si $p = [\alpha_1 \cdots \alpha_k]_1$ et $p' = [\alpha_1, \cdots, \alpha_k\alpha]_1$.
- $\mathrm{r}_{\mathrm{pop}_\alpha}(p, p')$ si et seulement si $p = [\alpha_1 \cdots \alpha_k\alpha]_1$ et $p' = [\alpha_1, \cdots, \alpha_k]_1$.
- $\mathrm{r}_\varepsilon(p, p')$ si et seulement si $p = p'$.

Pour les niveaux n supérieurs à 1, on introduit une opération de copie copy_n et son inverse $\overline{\mathrm{copy}}_n$, et on autorise les opérations de niveau inférieur qui s'appliqueront au sommet de la pile : $\mathrm{Ops}_n(\Gamma) = \{\mathrm{copy}_n, \overline{\mathrm{copy}}_n\} \cup \mathrm{Ops}_{n-1}(\Gamma)$. Ces opérations définissent les relations suivantes :

- $\mathrm{r}_{\mathrm{copy}_n}(p, p')$ si et seulement si $p = [p_1 \cdots p_k]_n$ et $p' = [p_1 \cdots p_k p_k]_n$.
- $\mathrm{r}_{\overline{\mathrm{copy}}_n}(p, p')$ si et seulement si $p = [p_1 \cdots p_k p_k]_n$ et $p' = [p_1 \cdots p_k]_n$.
- $\mathrm{r}_\theta(p, p')$, pour $\theta \in \mathrm{Ops}_{n-1}(\Gamma)$, si et seulement si $p = [p_1 \cdots p_k s]_n$, $p' = [p_1 \cdots p_k s']_n$ et $\mathrm{r}_\theta(s, s')$.

Toutes ces relations étant en fait des fonctions, on utilisera parfois, par soucis de concision, l'abus de notation suivant : pour toute opération θ et toutes n-piles p, p', on note $p' = \theta(p)$ si $\mathrm{r}_\theta(p, p')$.

On considérera enfin les mots d'opérations sur l'alphabet $\mathrm{Ops}_n(\Gamma)$. Un mot d'opérations $\theta = \theta_1 \cdots \theta_k$ sera simplement appelé opération, par soucis de simplicité, et si l'on veut spécifier qu'on utilise un élément de $\mathrm{Ops}_n(\Gamma)$ on parlera d'*opération de base*. Une opération définit une relation qui est simplement la composition des relations définies par les opérations de base la composant : $\mathrm{r}_\theta = \mathrm{r}_{\theta_k} \circ \cdots \circ \mathrm{r}_{\theta_1}$. Dit autrement, r est un morphisme de $\mathrm{Ops}_n(\Gamma)^*$ muni de la concaténation dans l'ensemble des relations binaires sur $\mathit{Stacks}_n(\Gamma)$ muni de la composition.

On définit enfin un morphisme sur les opérations, pour pouvoir manipuler leur opération inverse partielle (dans le sens où pour toute opération θ, $\theta \cdots \overline{\theta}$ définira toujours une relation incluse dans la relation identité). On a $\overline{\mathrm{push}_a} = \mathrm{pop}_a$, $\overline{\mathrm{pop}_a} = \mathrm{push}_a$, $\overline{\overline{\mathrm{copy}}_n} = \mathrm{copy}_n$ et pour $\theta = \theta_1 \cdots \theta_k$, $\overline{\theta} = \overline{\theta_k} \cdots \overline{\theta_1}$.

Nous présentons la forme faible des automates à pile d'ordre supérieur (c'est-à-dire avec uniquement des opérations simple sur chaque transition) et on impose que depuis un état donné, l'automate puisse emprunter uniquement des transitions étiquetées par des lettres ou uniquement des transitions étiquetées par ε. Un automate à pile d'ordre n est défini comme un uplet $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, F, \Delta)$ avec :

- Q un ensemble fini d'états, découpé entre les états observables Q_{Obs} et les états internes Q_{Int}.
- Σ un alphabet fini d'entrée.
- Γ un alphabet fini de piles.
- $q_0 \in Q_{\text{Obs}}$ l'état *initial*.
- $F \subseteq Q_{\text{Obs}}$ l'ensemble des états *finaux*.
- $\Delta \subseteq Q_{\text{Obs}} \times \text{Ops}_n(\Gamma) \times \Sigma \times Q \cup Q_{\text{Int}} \times \text{Ops}_n(\Gamma) \times \{\varepsilon\} \times Q$.

Une configuration de $\mathcal{A}$ est un couple (q, p) avec q un état de Q et $p \in \mathit{Stacks}_n(\mathit{Gamma})$ une n-pile. La configuration initiale de $\mathcal{A}$ est (q, ε), et l'ensemble des configurations finales est $F \times \mathit{Stacks}_n(\Gamma)$. L'automate à pile induit les relations $\xrightarrow[\mathcal{A}]{a}$ pour tout $a \in \Sigma \cup \{\varepsilon\}$ définies par : $(q, p) \xrightarrow[\mathcal{A}]{a} (q', p')$ si et seulement si il existe une transition de Δ, $\tau = (q, \theta, a, q')$ et $p' = \theta(p)$.

Les langages reconnus par les automates d'ordre n sont les langages indexés d'ordre n, ou le niveau n de la hiérarchie OI [Mas74, Mas76, Dam82]. De la même manière que les langages algébriques (qui sont le niveau 1 de cette hiérarchie), ces langages sont clos par union, mais ni par complément ni par intersection et sont strictement inclus dans les langages contextuels.

On peut de même définir le graphe associé à un automate à pile A d'ordre n, $\mathcal{G}_{\mathcal{A}} = (Q \times \mathit{Stacks}_n(\Gamma), \{((q, p), a, (q', p')) \mid (q, p) \xrightarrow[\mathcal{A}]{a} (q', p')\})$. De même que précédemment, on considérera la restriction de ces graphes par accessibilité à partir de la configuration initiale, à des ensembles reconnaissables de sommets (que nous définirons). On considérera également $\mathcal{G}_{\mathcal{A},T}$ le graphe des transitions de l'automate à pile d'ordre n qui est obtenu en ne conservant que les configurations contenant un état observable de l'automate à pile et en contractant les arcs étiqueté par ε. Formellement, on a $\mathcal{G}_{\mathcal{A},T} = (V, E)$ avec

- $V = \{(q, p) \mid q \in Q_{\text{Obs}} \wedge (q_0, []_2) \xrightarrow[\mathcal{A}]{*} (q, p)\}$,
- $E = \{((q_1, p_1), a, (q_2, p_2)) \mid (q_1, p_1) \xrightarrow[\mathcal{A}]{a\varepsilon^*} (q_2, p_2)\}$.

Exemple 3.7. *On considère l'automate à pile d'ordre* 2 $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{a, b, c\}, \{A, B\}, q_0, \{q_7\}\Delta)$*, avec*

$$\begin{aligned}\Delta = \{&(q_0, \text{push}_B, a, q_1)\\ &(q_1, \text{push}_A, \varepsilon, q_2)\\ &(q_1, \text{copy}_2, \varepsilon, q_3)\\ &(q_2, \text{push}_A, a, q_2)\\ &(q_2, \text{copy}_2, a, q_3)\\ &(q_3, \text{pop}_A, b, q_3)\\ &(q_3, \text{pop}_B, b, q_4)\\ &(q_4, \text{push}_B, \varepsilon, q_5)\\ &(q_5, \text{push}_A, \varepsilon, q_5)\\ &(q_5, \overline{\text{copy}}_2, \varepsilon, q_6)\\ &(q_6, \text{pop}_A, c, q_6)\\ &(q_6, \text{pop}_B, c, q_7)\}\end{aligned}$$

Les états observables sont q_0, q_2, q_3, q_6, q_7 et les états internes sont q_1, q_4, q_5.

Cet automate reconnaît le langage $L = \{a^n b^n c^n \mid n \geq 0\}$. En effet, l'automate commence par lire des a en empilant un symbole par lecture de a. Ensuite, il copie sa pile d'ordre 1*, puis lit des b en dépilant un symbole par lecture. Une fois la pile d'ordre* 1 *la plus haute vidée, il la remplit avec des symboles sans rien lire, puis la détruit. Enfin, il lit des c en dépilant un symbole par lecture, jusqu'à arriver à la pile vide qui est la seule pouvant apparaitre dans un état final.*

On représente à la figure 3.7 la restriction aux sommets accessibles depuis la configuration initiale $(q_0, []_2)$ du graphe associé à cet automate à piles d'ordre 2.

On représente à la figure 3.8 le graphe des transitions de cet automate (c'est-à-dire obtenu en contractant les arcs étiquetés par ε).

Ensembles reconnaissables de piles d'ordre supérieur

Dans [Car05], Carayol définit une notion de reconnaissabilité sur les éléments de $Stacks'_n(\Gamma)$ dans le but d'étendre la propriété 3.4 aux automates à piles d'ordre supérieur. Un ensemble de n-piles est dit reconnaissable si et seulement si il est l'ensemble des piles apparaissant dans les configurations finales accessibles d'un automate à pile d'ordre n. Comme un automate à pile d'ordre n dans lequel on ne considère pas la pile est un automate fini, les ensembles d'opérations qui apparaissent sur le chemin d'un état initial à un état final sont donc des ensembles reconnaissables d'opérations (vues comme des mots). De ce fait, un ensemble reconnaissable de n-piles est défini comme l'image de la pile vide par un ensemble reconnaissable d'opérations.

Définition 3.2. *Un ensemble de n-piles R est reconnaissable si et seulement si il existe un ensemble reconnaissable d'opérations $L \in \mathrm{Ops}_n(\Gamma)$ tel que $R = \{p \mid \exists \theta \in L, p = \theta([]_1)$.*

Cette définition permet d'étendre la proposition 3.4 aux automates à pile.

Proposition 3.7. *L'ensemble $\{p \mid \exists q \in F, (q_0, \varepsilon) \xrightarrow[\mathcal{A}]{*} (q, p)\}$ est un ensemble reconnaissable de n-piles.*

Représentation normalisée des ensembles reconnaissables d'opérations Un des problèmes de la définition de $\mathrm{Ops}_n(\Gamma)^*$ est qu'il existe des relations qui peuvent être définies par plusieurs opérations (en réalité, chaque relation peut être définie par une infinité d'opérations). Par exemple, les opérations pop_α, $\mathrm{push}_\beta \mathrm{pop}_\beta \mathrm{pop}_\alpha$ et $\mathrm{pop}_\alpha \mathrm{copy}_2 \overline{\mathrm{copy}}_2$ définissent toutes les trois la même relation. Cela est peu gênant si on se concentre sur des systèmes finis d'opérations, puisqu'on donne pour chaque relation l'un de ses représentants. Cependant, lorsque l'on va s'intéresser à des ensembles reconnaissables d'opérations, il sera pertinent d'avoir une représentation normalisée de ces ensembles reconnaissables, dans le but notamment de montrer que le problème d'accessibilité des automates à piles d'ordre n est décidable. Cette représentation reviendra à éliminer les «circuits» dans une opération, c'est à dire les sous-suites d'opérations définissant une relation incluse dans l'identité. Cela permettra de donner une représentation *constructive* et *minimale* des ensembles reconnaissables de piles, dans le sens où on élimine ces circuits. Les opérations ainsi obtenues sont dites *réduites* [Car06]. On peut en effet associer à tout couple de piles p, p' une unique opération réduite $\rho_{p,p'}$ telle que $\mathrm{r}_{\rho_{p,p'}}(p, p')$. Cependant, si on élimine seulement les circuits, on n'obtiendra pas des relations équivalentes. En effet, la relation définie par $\mathrm{pop}_\alpha \mathrm{push}_\alpha$ n'est pas la relation identité, mais la relation identité restreintes aux piles dont la plus haute lettre est α. Ainsi, en général, il n'existe pas d'ensemble d'opérations réduites définissant la même relations qu'un ensemble d'opérations quelconque.

Pour remédier à ce problème, Carayol étend $\mathrm{Ops}_n(\Gamma)$ pour y ajouter des opérations de test qui vérifient si la pile à laquelle elles s'appliquent est dans un certain langage reconnaissable de piles. Formellement, une opération de test est notée T_L avec $L \in \mathrm{Rec}(Stacks_n(\Gamma))$ et définit la relation $\mathrm{r}_{T_L} = \{(p, p) \mid p \in L\}$. On note $\mathbb{T}_n(\Gamma)$ l'ensemble des opérations de tests T_L avec L un ensemble reconnaissable de $Stacks_n(\Gamma)$. Une relation définie par un ensemble reconnaissable d'opérations de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))$ est appelée suffixe-reconnaissable d'ordre n. Cette dénomination [2] provient des travaux de Caucal qui définit les relations suffixe-reconnaissables sur les mots comme des relations de la forme $W(U \times V)$ où U, V et W sont des ensembles reconnaissables de mots. Carayol a ensuite étendu cette notion aux piles d'ordre supérieur en

2. dans [Car06], ces relations sont dites préfixe-reconnaissables d'ordre n.

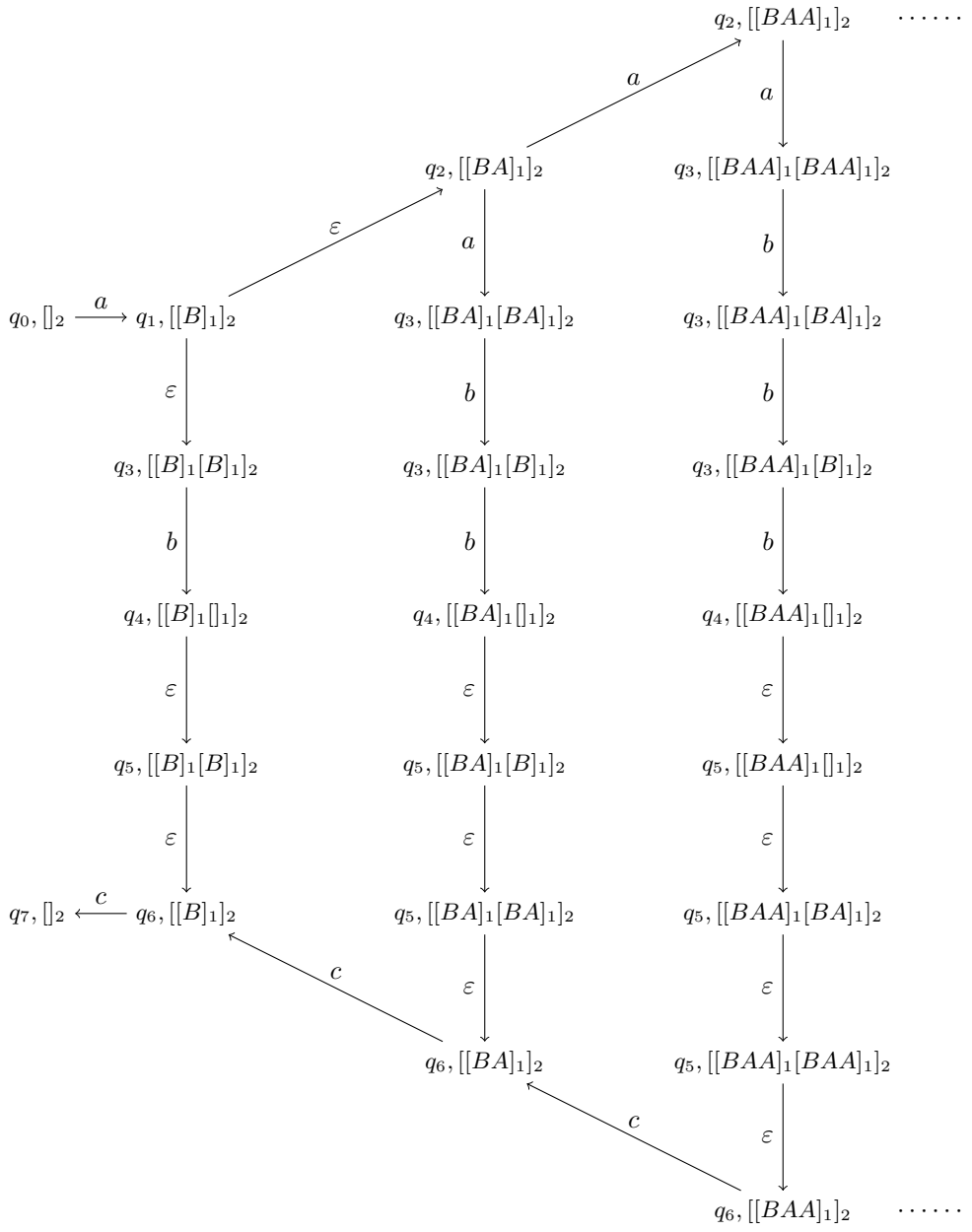


FIGURE 3.7 – Le graphe enraciné en $q_0, []_2$ de l'automate à pile d'ordre 2 de l'exemple 3.7

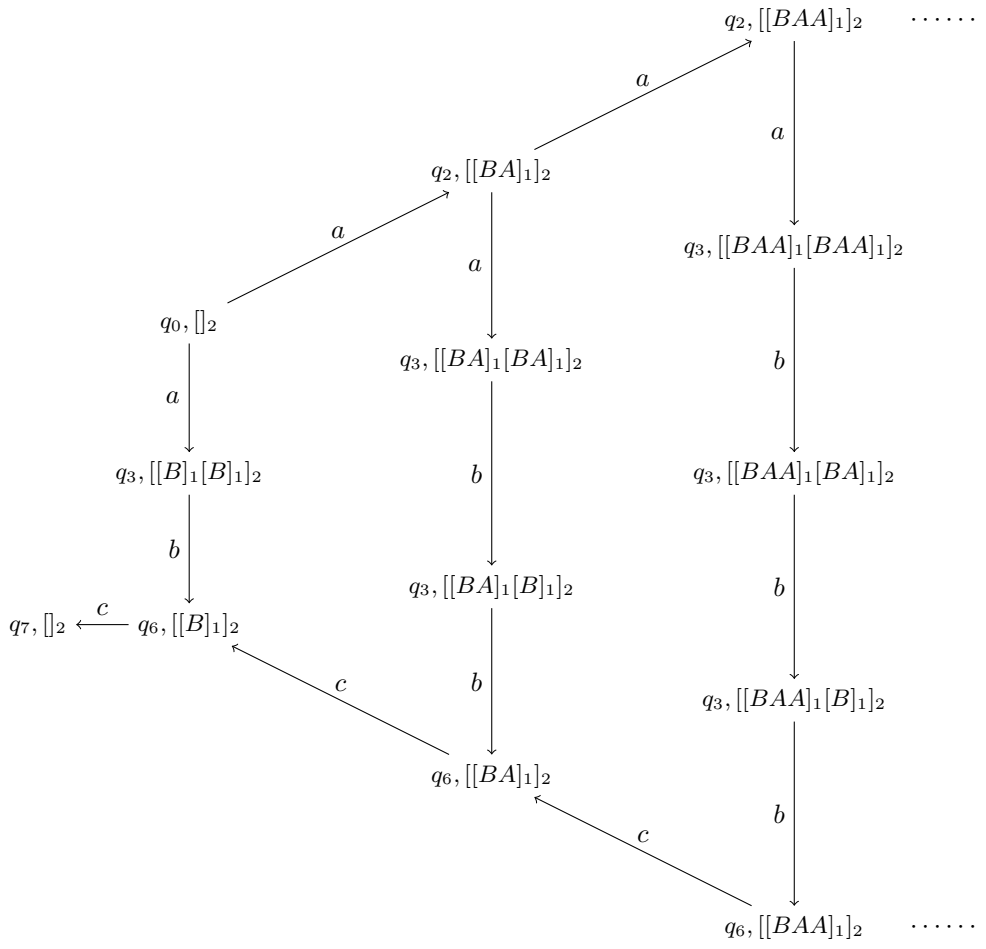


FIGURE 3.8 – Le graphe des transitions de l'automate à pile d'ordre 2 de l'exemple 3.7

observant qu'une relation de cette forme au niveau 1 peut s'écrire comme l'ensemble des opérations $\{\mathrm{pop}_{u_{|u|}} \cdots \mathrm{pop}_{u_1} T_W \mathrm{push}_{v_1} \cdots \mathrm{push}_{v_{|v|}} \mid u \in U, v \in V, u_1 \neq v_1\}$. Les relations suffixes-reconnaissables d'ordre n contiennent les relations définies par des ensembles reconnaissables d'opérations de $\mathrm{Ops}_n(\Gamma)$, et disposent d'une représentation normalisée.

Définition 3.3. *On définit inductivement les ensembles* $\mathrm{Rew}_n(\Gamma)$ *et* $\mathrm{Norm}_n(\Gamma)$ *:*

- $\mathrm{Norm}_1(\Gamma) = \mathrm{Rec}(\{\mathrm{push}_a \mid a \in \Gamma\}^*)$,
- $\mathrm{Rew}_1(\Gamma) = \{\overline{U} \cdot T_L \cdot V \mid U, V \in \mathrm{Norm}_1(\Gamma), L \in \mathrm{Rec}(\mathit{Stacks}_1(\Gamma)), \mathrm{First}(U) \cap \mathrm{First}(V) \subseteq \{\varepsilon\}\}$,
- $\mathrm{Norm}_n(\Gamma) = \mathrm{Rew}_{n-1}(\Gamma) \cdot \mathrm{Rec}((\mathrm{copy}_n \cdot \mathrm{Rew}_{n-1}(\Gamma))^*)$,
- $\mathrm{Rew}_n(\Gamma) = \{\overline{U} \cdot T_L \cdot V \mid U, V \in \mathrm{Norm}_n(\Gamma), L \in \mathrm{Rec}(\mathit{Stacks}_n(\Gamma)), \mathrm{First}(U) \cap \mathrm{First}(V) \subseteq \{\varepsilon\}, \mathrm{Last}(L) \cap (\mathrm{First}(U) \cup \mathrm{First}(V)) \subseteq \{\varepsilon\}\}$,

où, étant donné une opération ρ, $\mathrm{First}(\rho)$ *est la première opération de* $\mathrm{Ops}_1(\Gamma)$ *apparaissant dans* ρ*, et* $\mathrm{First}(\rho) = \varepsilon$ *si* ρ *ne contient pas d'opération de* $\mathrm{Ops}_1(\Gamma)$*. On étend cette notation aux ensembles d'opérations :* $\mathrm{First}(U) = \{\mathrm{First}(\rho) \mid \rho \in U\}$*, et aux langages :* $\mathrm{First}(L) = \{\mathrm{First}(\rho_{[]_n,p}) \mid p \in L\}$ *(où* $\rho_{[]_n,p}$ *est l'unique opération réduite transformant* $[]_n$ *en* p*). On définit* Last *de manière similaire (mais avec la dernière opération).*

Théorème 3.8 (4.5.12, [Car06])**.** *Pour toute ensemble d'opérations* U *de* $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))$*, il existe une union finie d'ensembles de* $\mathrm{Rew}_n(\Gamma)$ *définissant la même relation.*

Ce théorème est obtenu en montrant qu'il est toujours possible de transformer un automate sur $\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma)$ en un automate équivalent reconnaissant uniquement des suites de $\mathrm{Rew}_n(\Gamma)$. Un tel automate est dit *normalisé*. Cela permet également de montrer que les relations suffixe-reconnaissables forment une algèbre de Boole et sont closes par itération.

Graphes liés à des automates à pile d'ordre supérieur

Dans [Cau02], Caucal introduit une hiérarchie [3] de graphes que nous appelons ici suffixe-reconnaissables d'ordre supérieur comme une extension des graphes suffixe-reconnaissables, qui forment le premier niveau de la hiérarchie. Le niveau 0 de cette hiérarchie est constituée des graphes finis. Un graphe du niveau n est défini comme le résultat de l'application d'une substitution rationnelle inverse au dépliage d'un graphe du niveau $n-1$. Cette définition implique directement la décidabilité de la MSO-théorie des graphes de la hiérarchie, puisque comme nous l'avons vu, ces transformations préservent la décidabilité de MSO, et que la MSO-théorie des graphes finis est décidable. Tout comme au niveau 1, ces graphes disposent d'une variété de représentations dont nous citons les plus remarquables dans le théorème suivant, en les attribuant à leurs auteurs. Comme dans le cas des automates à pile, on élimine implicitement les sommets isolés.

Théorème 3.9. *Pour tout graphe* $\mathcal{G} = (V, E)$*, les propositions suivantes sont équivalentes, à isomorphisme près :*

1. *$\mathcal{G}$ est obtenu en appliquant une substitution rationnelle inverse au dépliage d'un graphe du niveau $n-1$ de la hiérarchie [Cau02].*
2. *$\mathcal{G}$ est obtenu par l'application successive de n opérations de structure arborescente suivie d'une interprétation monadique à un graphe fini [CW03].*
3. *$\mathcal{G}$ est le graphe des transitions de $\mathcal{A}$ un automate à pile d'ordre n, c'est à dire V est un ensemble reconnaissable de configurations et $E = \{((q,p), a, (q',p')) \mid (q,p) \xrightarrow[\mathcal{A}]{\varepsilon^* a \varepsilon^*} (q',p')\}$ [CW03].*
4. *Les arcs de $\mathcal{G}$ sont définis par des relations suffixe-reconnaissables d'ordre n : $V = \mathit{Stacks}_n(\Gamma)$, $E = \{(p, a, p') \mid (p, p') \in R_a\}$ où pour tout $a \in \Sigma$, R_a est une relation suffixe-reconnaissable d'ordre n [Car05].*

3. Cette hiérarchie est également appelée «hiérarchie à piles» par son auteur.

5. *$\mathcal{G}$ est la solution d'un VR-système d'équations représenté par un graphe suffixe-reconnaissable d'ordre $n-1$ [CC03].*

Puisque ce sont les graphes de transition d'un automate à pile d'ordre n, les langages reconnus entre deux ensembles reconnaissables de sommets de ces graphes sont les langages du niveau n de la hiérarchie OI.

Par ailleurs, dans [Cau02], Caucal montre que les arbres obtenus par dépliage d'un graphe du niveau n contiennent les solutions des schémas de programmes d'ordre supérieur sûrs [KNU02] et une hiérarche de termes définis par des substitutions de premier ordre itérées [CK02].

Récapitulons les propriétés que nous avons présentées sur les graphes suffixe-reconnaissables d'ordre n, en insistant sur les réponses aux problèmes présentés au début de ce chapitre.

- Ils disposent des représentations présentées au théorème précédent.
- Ils possèdent une MSO-théorie décidable.
- L'ensemble des sommets accessibles à partir d'un sommet donné est un ensemble reconnaissable de n-piles, et est donc calculable.
- Les langages reconnus par ces graphes sont les langages indexés d'ordre n (ou du niveau n de la hiérarchie OI).
- Cette classe de graphes est close par union, MSO-interprétation, restriction à des ensembles rationnels de sommets et substitution rationnelle inverse.
- Le graphe obtenu par l'application successive de $n-1$ opérations de structures arborescente à l'arbre binaire complet est un générateur de cette classe par MSO-interprétation.

3.2.3 Un modèle équivalent : les systèmes de réécriture de piles

Jusqu'ici, nous avons présenté les modèles classiques définissant les graphes de la hiérarchie suffixe-reconnaissable. Dans ce paragraphe, nous présentons un formalisme équivalent, que nous intitulerons «systèmes de réécriture de piles». Ce formalisme nous permettra d'avoir une continuité avec les systèmes de réécriture d'arbres de piles définis au chapitre 4, et de simplifier la preuve que ce formalisme est effectivement une extension des automates à piles. Dans la suite de ce document, quand on parlera de piles et de systèmes associés à des piles, nous ferons référence au formalisme que nous définissons ici. Pour simplifier notre propos, l'ensemble des piles de niveau n défini précédemment sera noté $\mathit{Stacks}'_n(\Gamma)$ et son ensemble d'opérations associé $\mathrm{Ops}'_n(\Gamma)$.

Nous commençons par modifier légèrement la définition des piles. On considère un alphabet fini Γ. On définit inductivement l'ensemble des piles d'ordre n comme suit :

- $\mathit{Stacks}_0(\Gamma) = \Gamma$.
- $\mathit{Stacks}_n(\Gamma) = \mathit{Stacks}_{n-1}(\Gamma)^+$.

Observons que la seule différence avec le formalisme présenté plus haut est l'absence de pile vide au niveau 1. Cela permet d'avoir une définition de piles qui est indépendante du niveau de la pile. Une notion similaire peut être trouvée dans [Eng91], où des alphabets de pile non nécessairement finis sont considérés, en particulier pour déterminer la complexité de problèmes de décision sur les automates à pile d'ordre supérieur (comme le problème du vide, par exemple). Bien que nous ne considérions pas des piles sur un ensemble quelconque, notre approche en est inspirée.

À cette définition de piles, on associe un ensemble d'opérations $\mathrm{Ops}_n(\Gamma)$. Encore une fois, il n'y aura plus de différence entre les opérations de niveau 1 et les opérations de niveau supérieur. Les opérations de niveau 0 permettront de manipuler les lettre en réécrivant une lettre en une autre. Les opérations à tous les autres niveaux permettront simplement de copier l'élément le plus haut d'une pile. On considère $\mathrm{Ops}_0(\Gamma) = \{\mathrm{rew}_{\alpha,\beta} \mid \alpha, \beta \in \Gamma\} \cup \{\varepsilon\}$, avec pour tout $\gamma, \delta \in \Gamma$, on a $\mathrm{r}_{\mathrm{rew}_{\alpha,\beta}}(\gamma, \delta)$ si et seulement si $\gamma = \alpha$ et $\delta = \beta$, et $\mathrm{r}_\varepsilon = \mathrm{id}$. On considère $\mathrm{Ops}_n(\Gamma) = \{\mathrm{copy}_n, \overline{\mathrm{copy}}_n\} \cup \mathrm{Ops}_{n-1}(\Gamma)$, avec pour toutes n-piles p, p', $\mathrm{r}_{\mathrm{copy}_n}(p, p')$ si et seulement si $p =$

$[p_1 \cdots p_k]_n$ et $p' = [p_1 \cdots p_k p_k]_n$, $\mathrm{r}_{\overline{\mathrm{copy}}_n}(p, p')$ si et seulement si $\mathrm{r}_{\mathrm{copy}_n}(p', p)$, et pour tout $\theta \in \mathrm{Ops}_{n-1}(\Gamma)$, $\mathrm{r}_\theta(p, p')$ si et seulement si $p = [p_1 \cdots p_k s]_n$, $p' = [p_1 \cdots p_k s']_n$ et $\mathrm{r}_\theta(s, s')$.

On définit maintenant la notion de système de réécriture de piles.

Définition 3.4. *Un système de réécriture de piles sur $Stacks_n(\Gamma)$ étiqueté par l'alphabet Σ est la donnée de $|\Sigma|$ ensembles R_a d'opérations de $\mathrm{Ops}_n(\Gamma)^*$.*

Si tous les R_a sont finis, le système de réécriture est dit fini.

Si tous les R_a sont des ensembles reconnaissables d'opérations, le système de réécriture est dit reconnaissable.

Les ensembles reconnaissables de piles induits par ces opérations sont identiques à ceux définis dans le modèle classique (à la présence de la pile vide près). De même que précédemment, on considère l'introduction d'opérations de test qui définissent la restriction de l'identité à un ensemble reconnaissables de n-piles. La notion de relation suffixe-reconnaissable et leur normalisation s'adaptent à ce formalisme en considérant que $\mathrm{Rew}_0 = \{\mathrm{rew}_{\alpha,\beta} \mid \alpha, \beta \in \Gamma\}$, et en appliquant la définition de Rew_n et de Norm_n à partir du niveau 1. Une relation suffixe-reconnaissable est ainsi la relation définie par un ensemble reconnaissable d'opérations de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$, et peut être définie par une union finie d'ensembles de Rew_n.

Dans ce formalisme, un graphe $\mathcal{G} = (V, E)$ est suffixe-reconnaissable d'ordre n si

- V est un ensemble reconnaissable de $Stacks_n(\Gamma)$,
- E est défini par des relations suffixe-reconnaissables définis par des ensemble reconnaissables d'opérations de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$.

Ces graphes suffixe-reconnaissables d'ordre n sont isomorphes aux graphes suffixe-reconnaissables d'ordre n présentés dans le formalisme précédent.

Théorème 3.10. *Les graphes suffixe-reconnaissables d'ordre n définis par par des ensemble reconnaissables d'opérations de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$ sont isomorphes aux graphes suffixes-reconnaissables d'ordre n définis par des ensemble reconnaissables d'opérations de $(\mathrm{Ops}'_n(\Gamma') \cup \mathbb{T}_n(\Gamma'))^*$, où Γ' est un alphabet fini.*

Pour montrer cela, il suffit de montrer que les relations définies par des ensembles reconnaissables d'opérations de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$ sont isomorphes aux relations définies par des ensembles reconnaissables d'opérations de $(\mathrm{Ops}'_n(\Gamma') \cup \mathbb{T}_n(\Gamma'))^*$. Pour cela, on montre que toute relation définie par une opération de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$ est isomorphe à la relation définie par un ensemble fini d'opérations de $(\mathrm{Ops}'_n(\Gamma') \cup \mathbb{T}_n(\Gamma'))^*$ et que toute relation définie par une opération de $(\mathrm{Ops}'_n(\Gamma') \cup \mathbb{T}_n(\Gamma'))^*$ est isomorphe à la relation définie par un ensemble fini d'opérations de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$. Puisque les modèles diffèrent uniquement par la définition des piles et des opérations de niveau 1, il suffit de montrer la propriété pour les relations définies au niveau 1, et plus précisément qu'il existe un morphisme injectif ϕ de $Stacks_1(\Gamma)$ vers $Stacks'_1(\Gamma')$, et que pour toute opération $\theta \in \mathrm{Ops}_1(\Gamma)$, il existe un ensemble d'opérations R_θ de $(\mathrm{Ops}'_1(\Gamma))^*$ telle que $\mathrm{r}_\theta(p, p')$ si et seulement si $\mathrm{r}_{R_\theta}(\phi(p), \phi(p'))$ et l'existence de morphismes injectifs dans l'autre direction.

Lemme 3.11. *Il existe $\phi : Stacks_1(\Gamma) \to Stacks'_1(\Gamma)$ et $\rho : \mathrm{Ops}_1(\Gamma) \cup \mathbb{T}_1(\Gamma) \to \mathcal{P}((\mathrm{Ops}'_1(\Gamma) \cup \mathbb{T}_1(\Gamma))^*)$ tels que pour tous $p, p' \in Stacks_1(\Gamma)$ et $\theta \in \mathrm{Ops}_1(\Gamma)$, on a $\mathrm{r}_\theta(p, p')$ si et seulement si $\mathrm{r}_{\rho(\theta)}(\phi(p), \phi(p'))$.*

Démonstration. On considère $\phi = \mathrm{id}$. Étant donné que $Stacks_1(\Gamma) \subseteq Stacks'_1(\Gamma)$, on définit ainsi une application injective. On définit ρ comme suit :

- $\rho(\varepsilon) = \{\varepsilon\}$.
- $\rho(\mathrm{rew}_{\alpha,\beta}) = \{\mathrm{pop}_\alpha \mathrm{push}_\beta\}$.
- $\rho(\mathrm{copy}_1) = \{\mathrm{pop}_\alpha \mathrm{push}_\alpha \mathrm{push}_\alpha \mid \alpha \in \Gamma\}$.
- $\rho(\overline{\mathrm{copy}}_1) = \{\mathrm{pop}_\alpha \mathrm{pop}_\alpha \mathrm{push}_\alpha \mid \alpha \in \Gamma\}$.

· $\rho(T_L) = T_L$

Pour $\theta = \varepsilon$, l'équivalence est immédiate.

Pour $\theta = \mathrm{rew}_{\alpha,\beta}$, on choisit p, p' telles que $\mathrm{r}_{\mathrm{rew}_{\alpha,\beta}}(p, p')$. Par définition on a $p = [p_1 \cdots p_k\alpha]_1$ et $p' = [p_1 \cdots p_k\beta]_1$. On a bien $\mathrm{r}_{\mathrm{pop}_\alpha \mathrm{push}_\beta}(p, p')$. L'autre direction s'obtient de la même manière.

Pour $\theta = \mathrm{copy}_1$, on choisit p, p' telles que $\mathrm{r}_{\mathrm{copy}_1}(p, p')$. Par définition, on a $p = [p_1 \cdots p_k]_1$ et $p' = [p_1 \cdots p_k p_k]_1$. On a donc, par définition, $\mathrm{r}_{\mathrm{pop}_{p_k}\mathrm{push}_{p_k}\mathrm{push}_{p_k}}(p, p')$. Inversement, si on a un α tel que $\mathrm{r}_{\mathrm{pop}_\alpha\mathrm{push}_\alpha\mathrm{push}_\alpha}(p, p')$, on a $p = [p_1 \cdots p_k\alpha]_1$ et $p' = [p_1 \cdots p_k\alpha\alpha]_1$, et donc $\mathrm{r}_{\mathrm{copy}_1}(p, p')$.

Pour $\theta = \overline{\mathrm{copy}}_1$ la démonstration étant très similaire à la précédente, nous la laissons au lecteur.

Pour $\theta = T_L$, l'équivalence est immédiate. □

Lemme 3.12. *Il existe $\phi : \mathit{Stacks}'_1(\Gamma) \to \mathit{Stacks}_1(\Gamma \cup \{\#\})$ où # est un symbole n'appartenant pas à Γ et $\rho : \mathrm{Ops}'_1(\Gamma) \cup \mathbb{T}_1(\Gamma) \to \mathcal{P}((\mathrm{Ops}_1(\Gamma) \cup \mathbb{T}_1(\Gamma))^*)$ tels que pour tous $p, p' \in \mathit{Stacks}_1(\Gamma)$ et $\theta \in \mathrm{Ops}_1(\Gamma)$, on a $\mathrm{r}_\theta(p, p')$ si et seulement si $\mathrm{r}_{\rho(\theta)}(\phi(p), \phi(p'))$.*

Démonstration. Dans $\mathit{Stacks}_1(\Gamma)$, la pile vide étant absente, on va simuler la pile vide par un symbole de fond de pile. Étant donnée une pile $p = [p_1 \cdots p_k]_1$, on définit $\phi(p) = [\#p_1 \cdots p_k]_1$. On définit ρ comme suit :

· $\rho(\varepsilon) = \varepsilon$.

· $\rho(\mathrm{pop}_\alpha) = \{\mathrm{rew}_{\alpha,\beta}\overline{\mathrm{copy}}_1 \mid \beta \in \Gamma \cup \{\#\}\}$.

· $\rho(\mathrm{push}_\alpha) = \{\mathrm{copy}_1\mathrm{rew}_{\beta,\alpha} \mid \beta \in \Gamma \cup \{\#\}\}$.

· $\rho(T_L) = T_{\phi(L)}$.

Pour $\theta = \varepsilon$, l'équivalence est immédiate.

Pour $\theta = \mathrm{pop}_\alpha$, on choisit p, p' telles que $\mathrm{rew}_{\mathrm{pop}_\alpha,}(p, p')$. Par définition, on a $p = [p_1 \cdots p_k\alpha]_1$ et $p' = [p_1 \cdots p_k]_1$. Donc $\phi(p) = [\#p_1 \cdots p_k\alpha]_1$ et $\phi(p') = [\#p_1 \cdots p_k]_1$, et on a bien $\mathrm{r}_{\mathrm{rew}_{\alpha,\mathrm{p_k}}\overline{\mathrm{copy}}_1}(\phi(p), \phi(p'))$ si $k \geq 1$ et $\mathrm{r}_{\mathrm{rew}_{\alpha,\#}\overline{\mathrm{copy}}_1}(\phi(p), \phi(p'))$ si $k = 0$. Supposons maintenant qu'on a p, p' et β tels que $\mathrm{rew}_{\mathrm{rew}_{\alpha,\beta}\overline{\mathrm{copy}}_1,}(\phi(p), \phi(p'))$. On a par définition $\phi(p) = [\#p_1 \cdots p_k\beta\alpha]_1$ et $\phi(p') = [\#p_1 \cdots p_k\beta]_1$ si $\beta \neq \#$ et $\phi(p) = [\#\alpha]_1$ et $\phi(p') = [\#]_1$ si $\beta = \#$. Ainsi on a $p = [p_1 \cdots p_k\beta\alpha]_1$ et $p' = [p_1 \cdots p_k\beta]_1$ ou $p = [\alpha]_1$ et $p' = []_1$. Dans tous les cas, on a bien $\mathrm{r}_{\mathrm{pop}_\alpha}(p, p')$.

Pour $\theta = \mathrm{push}_\alpha$, la démonstration étant très similaire à la précédente, nous la laissons au lecteur.

Pour $\theta = T_L$, l'équivalence est immédiate. □

On peut étendre le morphisme ρ pour envoyer les relations définies par des ensembles d'opérations de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$ vers les relations définies par des ensembles d'opérations de $(\mathrm{Ops}'_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$ (et vice versa). Ainsi, on obtient bien l'isomorphisme entre les relations définies par des ensembles reconnaissables d'opérations de $(\mathrm{Ops}_n(\Gamma) \cup \mathbb{T}_n(\Gamma))^*$ et celles définies par des ensembles reconnaissables d'opérations de $(\mathrm{Ops}'_n(\Gamma') \cup \mathbb{T}_n(\Gamma'))^*$. Et donc les graphes engendrés par des systèmes reconnaissables de piles d'ordre n sont isomorphes à ceux engendrés par des relations suffixe-reconnaissables d'ordre n.

Étant donné un alphabet fini Γ, on va isoler pour chaque ordre un générateur particulier des graphes suffixe-reconnaissables d'ordre n : l'application de n structures arborescentes au graphe fini dont les sommets sont les lettres de Γ et qui contient tous les arcs définis par les opérations $\mathrm{rew}_{\alpha,\beta}$. On appelle Δ^n_Γ le graphe obtenu par n application de l'opération de structure arborescente à ce graphe. Formellement, on définit inductivement Δ^n_Γ comme suit :

· $\Delta^0_\Gamma = (\Gamma, \{(\alpha, \mathrm{rew}_{\alpha,\beta}, \beta) \mid \alpha, \beta \in \Gamma\})$,

· $\Delta^n_\Gamma = \mathrm{Treegraph}_{\#_n}(\Delta^{n-1}_\Gamma)$.

On représente des fragments de Δ^1_Γ à la figure 3.9 et des fragments de Δ^2_Γ à la figure 3.10, avec $\Gamma = \{\alpha, \beta\}$. Informellement, les arcs de niveau 0 correspondent aux opérations de niveau 0, et les arcs étiquetés par $\#_i$ correspondent à l'opération copy_i. Il est ensuite aisé de décrire n'importe quel opération de niveau par un chemin (non-orienté) dans Δ^n_Γ.

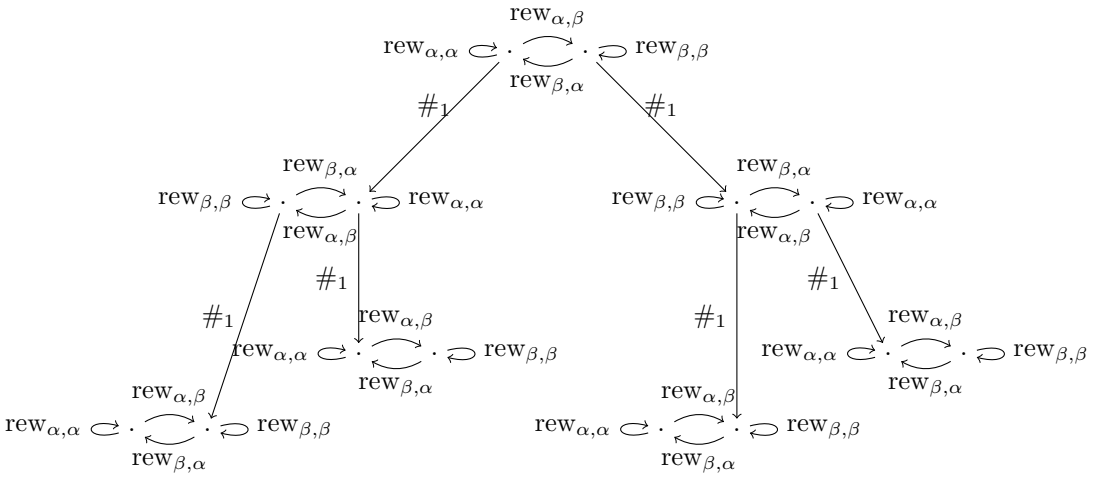


FIGURE 3.9 – Δ^1_Γ

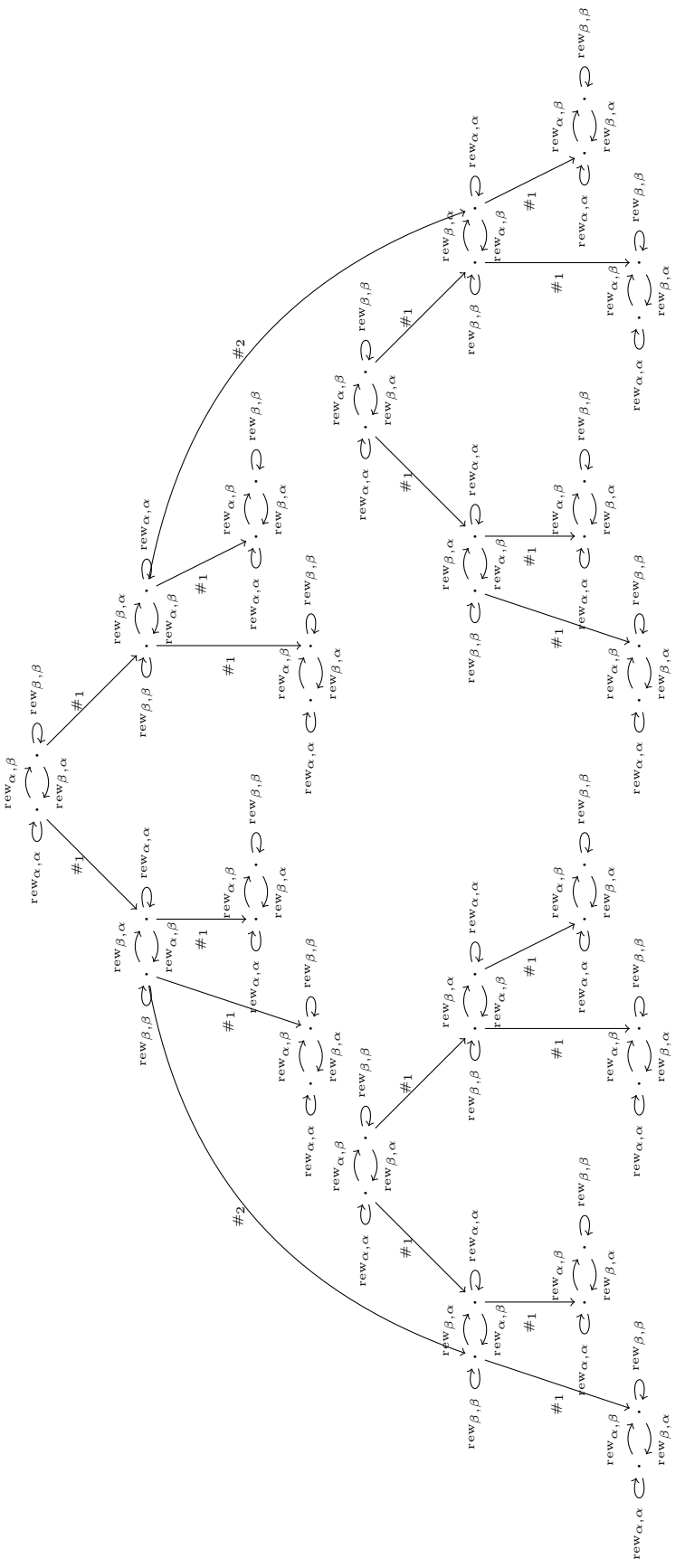


FIGURE 3.10 – Δ^2_Γ

3.3 Graphes liés aux arbres

On s'intéresse maintenant à des graphes dont la représentation interne est liée à des arbres et non plus à des mots ou des piles. On commence par définir formellement les arbres, les systèmes de réécriture suffixe d'arbres et les automates finis d'arbres, qui sont, à l'instar des systèmes de réécriture suffixe de mots/piles et des automates finis de mots, des éléments centraux des représentations internes des graphes qui nous intéressent ici. Nous définissons ensuite la notion de relation arbre-synchronisée sur des arbres et les graphes associés, appelés arbre-automatiques. Nous considérons également la notion de transducteur suffixe d'arbres (*ground tree transducers*, ou GTT) introduits par Dauchet, Tison et al. dans [DHLT90], qui définissent des relations incluses dans les relations arbre-synchronisées et rappelons les résultats liés à cette notion. Puis nous nous intéressons aux graphes de réécriture suffixe d'arbres, et aux langages qu'ils reconnaissent. Enfin, nous nous intéresserons aux classes mal connues des graphes arbre-automatiques d'ordre supérieur.

3.3.1 Arbres

Les arbres peuvent désigner une grande variété d'objets. Ceux que nous présentons ici sont vus comme une extension des mots, où chaque lettre au lieu d'avoir un unique successeur (ou aucun) peut en posséder plusieurs. Cette notion est liée à celle de *terme* dont elle diffère par la non-coordination d'un symbole et d'une arité : tout nœud de l'arbre peut être étiqueté par n'importe quel symbole de l'alphabet (alors que, par exemple, dans le cas des termes définissant une formule arithmétique, le symbole + possède obligatoirement deux fils). Cette distinction mise à part, les résultats que nous considérerons seront valables aussi bien sur les arbres que sur les termes. Aussi, nous prendrons le cas des arbres pour avoir un modèle plus simple.

On choisit un alphabet fini Γ et on considère l'ensemble $\mathcal{T}(\Gamma)$ des arbres étiquetés par Γ. Cet ensemble est défini comme suit.

Définition 3.5. *Un arbre d'arité au plus $d \in \mathbb{N}$ étiqueté par un ensemble Γ est une fonction t de $\{1, \cdots, d\}^*$ vers Γ telle que :*

- $\mathrm{Def}(t) \neq \emptyset$.
- $\mathrm{Def}(t)$ *est clos par préfixe.*
- $\forall u \in \{1, \cdots, d\}^*, i \in \{1, \cdots, d\}, u.i \in \mathrm{Def}(t) \Rightarrow \forall j \in \{1, \cdots, i\}, u.j \in \mathrm{Def}(t)$.

On note $\mathcal{T}_d(\Gamma)$ l'ensemble des arbres finis d'arité au plus d étiquetés par Γ. Quand il n'y a pas d'ambiguïté sur d, on note simplement $\mathcal{T}(\Gamma)$. Dans la suite, on considérera des arbres d'arité au plus 2, pour simplifier la présentation. Cela dit, ce qu'on présentera est également valable pour toute arité maximale d.

Étant donné un arbre t étiqueté par Γ, un élément de $\mathrm{Def}(t)$ est appelé un *nœud* de t.

Étant donnés deux nœuds u et v de t, on dit que :

- u est le *père* de v et v est le $i^{\text{ème}}$ *fils* de u si $v = u.i$.
- u est un *ancêtre* de v et v est un *descendant* de u si $u \sqsubseteq v$.
- u est une *feuille* de t si il n'a pas de descendant, sinon on dit que c'est un *nœud interne* de t. On note $\mathrm{fr}(t)$ la frontière de t, l'ensemble des feuilles de t.

Étant donnés t, t' deux arbres étiquetés par Γ et u une feuille de t, on définit $t[t']_u$ l'arbre de domaine $\mathrm{Def}(t) \cup u.\mathrm{Def}(t')$ tel que :

- $t[t']_u(u.v) = t'(v)$.
- $t[t']_u(v) = t(v)$ si $u \not\sqsubseteq v$.

Étant donné t un arbre étiqueté par Γ, u une de ses feuilles, et $\alpha \in \Gamma$, on note $t[\alpha]_u$ l'arbre obtenu en remplaçant l'étiquette de u dans t par α.

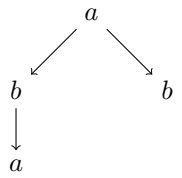


FIGURE 3.11 – Un arbre

Nous étendons également cette notation à des remplacement de plusieurs sous-arbres disjoints : étant donnés un entier k, un arbre C, $u_1, \cdots, u_k$ des feuilles de C et $t_1, \cdots, t_k$ des arbres, on définit récursivement $C[t_1, \cdots, t_k]_{u_1,\cdots,u_k} = (C[t_2, \cdots, t_k]_{u_2,\cdots,u_k})[t_1]_{u_1}$.

Faute de meilleure notation, dans le cas d'arbres relativement petits, nous pourrons donner une description explicite de l'arbre comme une fonction. Par exemple, on décrira l'arbre représenté à la figure 3.11 par $\{\varepsilon \to a, 1 \to b, 11 \to a, 2 \to b\}$.

3.3.2 Systèmes de réécriture d'arbres

On définit maintenant la notion de réécriture suffixe d'arbres. Elle est liée à la notion de réécriture sur les mots, étant donné qu'un mot peut être vu comme un arbre d'arité 1 (à la différence près que l'arbre vide n'existe pas). On peut également la voir, comme une extension des systèmes de réécriture de piles d'ordre 1 (et cette fois-ci, les 1-piles sont exactement les arbres d'arité 1, puisque dans le modèle que l'on a défini, la pile vide n'existe pas). Une *règle de réécriture* d'arbres est un couple d'arbres (g, d). La relation qu'elle définie est $\mathrm{r}_{(g,d)} = \{(t, t') \mid \exists C \in \mathcal{T}(\Gamma), u \in \mathrm{fr}(C), t = C[g]_u \wedge C[d]_u\}$, c'est-à-dire que deux arbres t et t' sont en relation par la règle (g, d) si t' peut être obtenu en remplaçant une occurrence de g par d dans t, à la condition que g soit un sous-arbre suffixe de t (c'est-à-dire qu'il n'y ait rien «en-dessous»). Notons que si l'on se restreint au cas des arbres d'arité 1, l'application d'une règle est bien la même que celle d'une règle de réécriture de mots (la seule différence étant que l'arbre vide n'existe pas).

Définition 3.6. *Un système de réécriture suffixe d'arbres R étiqueté par un alphabet Σ est la donnée de $|\Sigma|$ ensembles R_a de règles de réécriture suffixe d'arbres. Il existe éventuellement un ensemble R_ε de règles silencieuses étiquetées par $\varepsilon \notin \Sigma$. Il induit le graphe $\mathcal{G}_R = (\mathcal{T}(\Gamma), E)$, avec $E = \{(t, a, t') \mid \exists (g, d) \in R_a, \mathrm{r}_{(g,d)}(t, t')\}$.*

Exemple 3.8. *On considère les alphabets $\Gamma = \{\alpha, \beta\}$ et $\Sigma = \{a, b\}$. On définit le système de réécriture suffixe d'arbres sur $\mathcal{T}(\Gamma)$ étiqueté par Σ, $R_a = \{(\{\varepsilon \to \alpha\}, \{\varepsilon \to \alpha, 1 \to \alpha\})\}$ et $R_b = \{(\{\varepsilon \to \beta\}, \{\varepsilon \to \beta, 1 \to \beta\})\}$. On représente le graphe induit par ce système de réécriture restreint aux sommets accessibles depuis l'arbre $\{\varepsilon \to \alpha, 1 \to \alpha, 2 \to \beta\}$ en figure 3.12. Ce graphe est la grille infinie.*

3.3.3 Automates d'arbres

Comme on l'a vu les arbres sont une extension des mots dans le sens où un mot peut être vu comme un arbre d'arité 1. De même que les ensembles reconnaissables de mots sont liés à une notion d'automates de mots, il existe une notion d'ensembles reconnaissables d'arbres qui est liée à une notion d'automates d'arbres. Cette notion a été introduite par Doner [Don65, Don70] et par Thatcher et Wright [TW65, TW68], dans le but de prouver la décidabilité de la logique monadique du second ordre faible avec successeurs multiples (ou WSkS). Ces définitions originelles sont basées sur de l'algèbre universelle et sur la théorie des catégories (qu'il ne nous appartient pas de détailler ici). Ces automates d'arbres ont été étudiés entre la fin des années 60 et les années 70, notamment dans [EW67, Tha70, Bra68, Bra69, AG68]. Pour un catalogue des connaissances sur les automates d'arbres, on peut se référer au livre de Gécseg et Steinby *Tree Automata* [GS84],

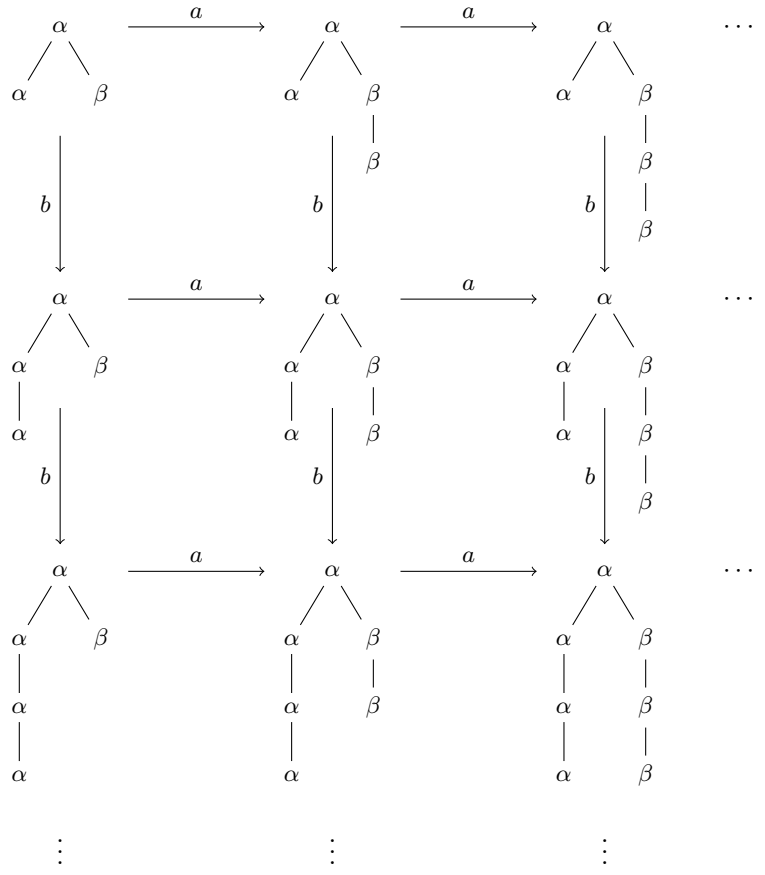


FIGURE 3.12 – Le graphe induit par un système de réécriture d'arbres.

ou au livre plus récent *Tree Automata, Techniques and Applications* [CDG$^+$07],. La plupart des résultats que nous citons y sont également référencés, et les définitions que nous donnons sont inspirées de celles qui y sont présentées. Nous choisissons de présenter la version descendante des automates d'arbres (introduite par Rabin dans [Rab69]). La définition des automates d'arbres originelle est la version descendante, mais les langages d'arbres reconnus par ces deux modèles étant égaux, et le passage de l'un à l'autre s'effectuant simplement en inversant les transitions, nous n'en parlerons pas formellement. Le lecteur est renvoyé vers [CDG$^+$07] pour plus de détails sur les différentes versions d'automates d'arbres. Notons toutefois que la principale différence entre les automates montants et descendants est que dans le cas des automates montants, la version déterministe de ces automates reconnaît les mêmes langages que la version non déterministe, alors que dans le cas des automates descendants, cela est faux. Dans le cadre de ce travail, nous n'avons pas besoin de déterminiser les automates d'arbres, aussi, nous pouvons sans risque nous restreindre au cas descendant.

Définition 3.7. *Un automate d'arbres descendant est un tuple $\mathcal{A} = (Q, i, \Delta)$ avec :*

- *Q un ensemble fini d'états.*
- *$i \in Q$ un état initial.*
- *$\Delta \subseteq Q \times (\bigcup_{0 \leq i \leq d} \Gamma \times Q^i)$ l'ensemble des transitions.*

Les transitions d'un automates d'arbres sont de la forme (q, a), $(q, (a, q'))$ et $(q, (a, q', q''))$, avec $a \in \Gamma$ et $q, q', q'' \in Q$. Une configuration d'un automate d'arbres est un arbre de $\mathcal{T}(\Gamma \cup Q)$ où les états n'apparaissent qu'aux feuilles. La configuration initiale est l'arbre contenant une seule feuille étiquetée par l'état initial. Étant donnés une transition τ et deux arbres t, t', on a $t \xrightarrow{\tau} t'$ si :

- Il existe $C \in \mathcal{T}(\Gamma \cup Q)$, $t = C[q]_u$ et $t' = C[a]_u$ si $\tau = (q, a)$.
- Il existe $C \in \mathcal{T}(\Gamma \cup Q)$, $t = C[q]_u$ et $t' = C[\{\varepsilon \to a, 1 \to q'\}]_u$ si $\tau = (q, (a, q'))$.
- Il existe $C \in \mathcal{T}(\Gamma \cup Q)$, $t = C[q]_u$ et $t' = C[\{\varepsilon \to a, 1 \to q', 2 \to q''\}]_u$ si $\tau = (q, (a, q', q''))$.

Le langage d'arbres accepté par l'automate est l'ensemble des arbres de $\mathcal{T}(\Gamma)$ accessibles à partir de la configuration initiale.

La version montante des automates d'arbres est simplement obtenue en inversant les transitions de Δ, en appelant i l'état final, et en considérant qu'un arbre est accepté si la configuration finale est accessible à partir de cet arbre.

Les automates finis d'arbres conservent la plupart des propriétés intéressantes des automates finis de mots. Comme nous l'avons déjà noté, la notion d'automate déterministe existe, mais n'est équivalente que dans le cas des automates montants (ceci dit, pour les automates descendants, il est toujours possible de trouver un automate descendant co-déterministe équivalent). Une autre propriété conservée est l'existence d'un automate minimal en nombre d'états unique reconnaissant le même langage qu'un automate donné ([Bra68, Bra69, AG68, EW67]). De même, de nombreux problèmes restent décidables, comme l'appartenance d'un arbre à un langage reconnaissable, la vacuité d'un langage, la finitude d'un langage, la vacuité du complémentaire d'un langage ou l'équivalence de deux automates. Plus de détails sur ces problèmes peuvent être trouvés dans [CDG$^+$07]. Enfin, la propriété qui nous intéresse le plus dans le cadre de notre travail est que les langages reconnaissables d'arbres forment une algèbre de Boole.

La notion d'automate d'arbres peut être vue comme un cas particulier d'un système de réécriture d'arbres, auquel on a adjoint un arbre initial et un ensemble d'arbres finaux (ceux n'ayant aucun état) : on peut observer facilement qu'un automate d'arbres est un système de réécriture suffixe d'arbres sur $\mathcal{T}(\Gamma \cup Q)$, puisque les transitions de la forme (q, a), $(q, (a, q'))$, $(q, (a, q', q''))$ définissent respectivement les mêmes relations que les règles de réécritures $(\{\varepsilon \to q\}, \{\varepsilon \to a\})$, $(\{\varepsilon \to q\}, \{\varepsilon \to a, 1 \to q'\})$, $(\{\varepsilon \to q\}, \{\varepsilon \to a, 1 \to q', 2 \to q''\})$.

Tout comme dans le cas des systèmes des automates à pile, la notion de reconnaissabilité est liée à l'ensemble de configurations accessibles dans un système de réécriture suffixe d'arbres à partir d'une configuration initiale, et la proposition 3.4 peut s'étendre ainsi :

Proposition 3.13 ([Bra69])**.** *Étant donné un système de réécriture suffixe d'arbres $\mathcal{R}$ et un arbre t_{Init}, l'ensemble des arbres accessibles dans $\mathcal{R}$ à partir de t_{Init}, $\mathcal{T}(R, t_{\text{Init}})$ est un ensemble reconnaissable d'arbres.*

3.3.4 Relations arbre-synchronisées

Si l'on souhaite décider le problème d'accessibilité d'un système de réécriture suffixe d'arbres, il faut être capable de décrire la clôture transitive d'un système de réécriture. Dans le cas des mots, les systèmes reconnaissables de réécriture suffixe de mots suffisent à la décrire. Dans le cas des arbres, ce n'est plus vrai, puisqu'un système reconnaissable de réécriture d'arbres impose de réécrire en un pas un seul sous-arbre, alors qu'en plusieurs pas, il est possible de réécrire plusieurs sous-arbres disjoints. Il est donc nécessaire de définir une autre notion de relations reconnaissables. Une telle notion peut être obtenue à partir de la notion d'automates d'arbres en superposant les arbres et en prenant des langages reconnaissables de couples d'arbres.

Définition 3.8. *Étant donnés k arbres $t_1, \ldots, t_k$ de $\mathcal{T}(\Gamma)$, on définit leur* superposition *comme l'arbre $\diamond(t_1, \ldots, t_k)$ de $\mathcal{T}((\Gamma \cup \{\Box\})^k)$ (où $\Box$ n'est pas un élément de Γ) tel que $\text{dom}(\diamond(t_1, \ldots, t_k) = \text{dom}(t_1) \cup \cdots \cup \text{dom}(t_k)$ et pour tout $u \in \text{dom}(\diamond(t_1, \ldots, t_k)$, $\diamond(t_1, \ldots, t_k)(u) = (\eta_1, \ldots, \eta_k)$, avec pour tout i, $\eta_i = t_i(u)$ si $u \in \text{dom}(t_i)$ et $\eta_i = \Box$ sinon.*

Informellement, la superposition des arbres $t_1, \ldots, t_k$ est l'arbre ayant pour domaine le domaine minimal contenant tous les domaines des t_i étiqueté par les uplets d'étiquettes des t_i, où les trous dans certains arbres sont remplacés par le symbole $\Box$.

Définition 3.9. *Une relation R sur $\mathcal{T}(\Gamma)^k$ est arbre-synchronisée si il existe un automate $\mathcal{A}$ sur $\mathcal{T}((\Gamma \cup \{\Box\})^k)$ tel que pour tous arbres $t_1, \ldots, t_k$ de $\mathcal{T}(\Gamma)$, $R(t_1, \ldots, t_k)$ si et seulement si $\diamond(t_1, \ldots, t_k) \in \mathcal{L}(\mathcal{A})$.*

Étant donné que les langages reconnaissables d'arbres forment une algèbre de Boole, c'est également le cas des relations arbre-synchronisées d'arbres. Ces relations contiennent bien les itérations des systèmes de réécriture d'arbres. Cependant, ces relations ne sont pas closes par itération. La propriété suivante résume ces résultats.

Proposition 3.14 ([DT90])**.** *Les relation arbre-synchronisées sont closes par union, intersection, complémentaire, projection et cylindrification, mais pas par itération.*

On peut définir de manière analogue les relations mot-synchronisées comme la restriction des relations arbre-synchronisées aux mots (vus comme des arbres d'arité 1). Elles possèdent les mêmes propriétés, notamment de clôture. L'exemple suivant montre une relation mot-synchronisée qui n'est pas close par itération.

Exemple 3.9. *La relation $R = \{(a^n b^m, a^{n-1} b^{m-1}) \mid n, m \geq 0\}$ est une relation mot-synchronisée : elle est reconnue par l'automate fini sur $(\Gamma \cup \{\Box\})^2$ représenté à la figure 3.13. Cependant, son itération $R^* = \{(a^n b^m, a^{n-k} b^{m-k}) \mid n, m \geq 0, k \leq \min(n, m)\}$ n'est pas mot-synchronisée, puisqu'il faudrait que l'automate fini la reconnaissant soit capable de compter, ce qui est impossible.*

3.3.5 Graphes arbre-automatiques

Un graphe arbre-automatique est un graphe dont les arcs sont définis par des relations arbre-synchronisées binaires. Étant données des relations arbre-synchronisées binaires R_a, pour a dans un alphabet fini Σ, son graphe associé est $\mathcal{G}_R = (\mathcal{T}(\Gamma), \{(t, a, t') \mid (t, t') \in R_a\})$, dans lequel on élimine implicitement les sommets isolés.

Les graphes automatiques disposent des représentations suivantes :

Théorème 3.15. *Étant donné un graphe $\mathcal{G}$, les propositions suivantes sont équivalentes, à isomorphisme près :*

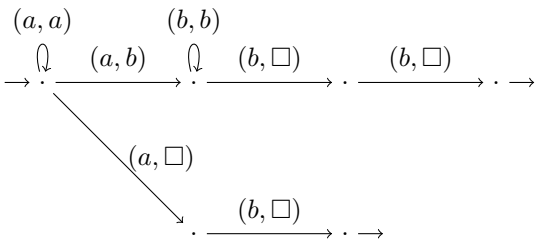


FIGURE 3.13 – L'automate reconnaissant la relation $\{(a^n b^m, a^{n-1} b^{m-1}) \mid n, m \geq 0\}$.

1. *$\mathcal{G}$ est arbre-automatique [DT90].*
2. *$\mathcal{G}$ est obtenu en appliquant une interprétation à ensembles finis à un graphe suffixe-reconnaissable d'ordre 1 [CL07].*
3. *$\mathcal{G}$ est VRS-équationnel [Col04].*

Théorème 3.16 ([DT90])**.** *Les graphes arbre-automatiques possèdent une* FO*-théorie décidable.*

Ce résultat a dans un premier temps été obtenu par Dauchet et Tison dans [DT90]. Cependant, il est aussi une conséquence directe de la représentation 2 du théorème 3.15 puisque les graphes suffixe-reconnaissables possèdent une MSO-théorie décidable.

Les graphes dont les arcs sont définis par des relations mots-synchronisées sont appelés graphes automatiques. Ces graphes possèdent des propriétés similaires à celles des graphes arbre-automatiques et ne sont pas cruciaux dans notre étude, aussi, nous n'en parlerons pas plus. Le lecteur intéressé par ces graphes particuliers peut lire [Hod83, KN95, BG00] pour plus de détails.

Récapitulons les propriétés que nous avons présentées sur les graphes arbre-automatique, en insistant sur les réponses aux problèmes présentés au début de ce chapitre :

- Ils disposent des représentations présentées au théorème précédent.
- Ils possèdent une FO-théorie décidable.
- L'ensemble des sommets accessibles à partir d'un sommet donné n'est pas calculable [Löd02].
- Les langages reconnus par ces graphes sont les langages de ETIME [Mey08].
- Cette classe de graphes est close par union, FO-interprétation, restriction à des ensembles rationnels de sommets.

3.3.6 Systèmes reconnaissables de réécriture suffixe d'arbres

Une des difficultés de la notion de relations arbre-synchronisées est, comme nous l'avons déjà remarqué, qu'elles ne sont pas closes par itération. La notion de transducteur suffixe d'arbres (ground tree transducer en anglais, ou GTT) a été introduite dans [DTHL87, DHLT90] dans le but de prouver la confluence des systèmes de réécriture suffixe d'arbres. Il forment une sous-classe stricte des relations reconnaissables d'arbres et contiennent les clôtures par itérations des systèmes de réécriture d'arbres. Ils sont donc essentiels pour prouver que les graphes des systèmes de réécriture d'arbres possèdent une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable. Dans la définition originelle de [DTHL87], un transducteur suffixe d'arbres est défini comme un couple d'automates d'arbres. On présente ici une vision équivalente en s'affranchissant des automates pour donner directement les ensembles reconnaissables d'arbres reconnus.

Définition 3.10. *Un transducteur suffixe d'arbres sur $\mathcal{T}(\Gamma)$ étiqueté par Σ est la donnée de $|\Sigma|$ ensembles $\mathcal{R}_a$ de règles qui sont des unions finies d'ensembles de la forme $U \times V$ avec U et V des ensembles reconnaissables d'arbres.*

Étant donnés deux arbres t, t' et une lettre a, on a $t \xrightarrow{a} t'$ si il existe k couples d'arbres $(g_1, d_1), \cdots, (g_k, d_k) \in R_a$, un arbre C et k feuilles $u_1, \cdots, u_k$ de C tels que $t = C[g_1, \cdots, g_k]_{u_1,\cdots,u_k}$ et $t' = C[d_1, \cdots, d_k]_{u_1,\cdots,u_k}$. Observons que la différence entre la relation reconnue par un transducteur suffixe d'arbres et celle définie par un système reconnaissable de réécriture suffixe d'arbres est que dans le cas du transducteur, on effectue un nombre arbitraire de réécriture parallèle, alors que dans le cas du système de réécriture, on n'en effectue qu'une seule. Cela est nécessaire pour attraper l'itération d'un système de réécriture d'arbres, puisque quand on effectue plusieurs pas dans un système de réécriture, on peut effectuer des réécriture à différents endroits de l'arbre. Cette difficulté n'apparaissait pas dans le cas des mots, puisqu'il n'y a qu'un seul endroit où effectuer une réécriture suffixe dans un mot. La notion de transducteur suffixe d'arbres est suffisante pour contourner ce problème.

Proposition 3.17 ([DT90])**.** *La clôture transitive d'une relation définie par un système de réécriture suffixe d'arbres peut être définie par un transducteur suffixe d'arbres.*

La clôture transitive d'une relation définie par un transducteur suffixe d'arbres peut être définie par un transducteur suffixe d'arbres.

Notons que contrairement au cas des mots, la clôture transitive des relations définies par des systèmes de réécriture d'arbres ne sont pas les systèmes reconnaissables de réécriture d'arbres. Cela est dû au fait que dans les systèmes de réécriture, on ne réécrit qu'un seul sous-arbre dans l'application d'une règle, même si on autorise des règles qui sont des couples d'ensembles reconnaissables d'arbres. Le transducteur suffixe d'arbres contourne cette difficulté en réécrivant n'importe quel nombre de sous-arbres de manière simultanée (et c'est la seule différence de définition), rattrapant ainsi la clôture transitive des systèmes de réécriture suffixe d'arbres.

3.3.7 Graphes de systèmes de réécriture d'arbres

De manière analogue aux graphes associés à des automates à piles, on peut définir les graphes associés aux systèmes de réécriture d'arbres : étant donné R un système de réécriture d'arbres (fini ou reconnaissable), on définit son graphe associé $\mathcal{G}_R = (\mathcal{T}(\Gamma), E)$ avec $E = \{(t, a, t') \mid \exists (g, d) \in R_a, \mathrm{r}_{(g,d)}(t, t')\}$. On définit également de manière analogue les graphes associés à des transducteurs suffixes d'arbres. Les restrictions des graphes des systèmes de réécriture d'arbres à des ensembles de sommets accessibles dans ce système de réécriture à partir d'un ensemble reconnaissable d'arbres disposent des représentations suivantes :

Théorème 3.18. *Étant donné un graphe $\mathcal{G}$, les affirmations suivantes sont équivalentes :*

1. *Il existe L un ensemble reconnaissable d'arbres et R un système reconnaissable de réécriture suffixe d'arbres tels que $\mathcal{G}$ est isomorphe à la restriction à l'ensemble des sommets accessibles dans R depuis les sommets de L de $\mathcal{G}_R$.*
2. *$\mathcal{G}$ est VRA-équationnel [Col02].*

Ces représentations ne donnent pas directement la décidabilité d'une théorie logique sur ces graphes. Un tel résultat a cependant été obtenu par Dauchet et Tison :

Théorème 3.19 ([DT90])**.** *Les graphes associé à des systèmes de réécriture suffixe d'arbres ont une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable.*

La preuve de ce théorème repose sur deux caractéristiques des systèmes de réécriture suffixe d'arbres. La première étant que la clôture transitive d'un système de réécriture suffixe d'arbres est un transducteur suffixe d'arbres, et donc que le graphe d'un système de réécriture suffixe d'arbres auquel on ajoute les arcs définis par sa clôture transitive est isomorphe à un graphe dont les arcs sont définis soit par un système de réécriture suffixe d'arbres, soit par un transducteur suffixe d'arbres. La seconde est que les systèmes de réécriture suffixe d'arbres et les transducteurs suffixes d'arbres sont inclus dans les relations arbre-synchronisées, définies par

des langages reconnaissables de couples d'arbres, qui possèdent une FO-théorie décidable, ce qui est donc également le cas des systèmes de réécriture suffixe d'arbres et des transducteurs suffixe d'arbres. La réunion de ces deux ingrédients permet de déduire ce théorème. De plus, cela permet également de montrer le même théorème pour les graphes des transducteurs suffixes d'arbres, puisque la clôture transitive de ces derniers est également isomorphe au graphe d'un transducteur suffixe d'arbres. Cependant, ils ne disposent pas d'une autre représentation connue que leur représentation interne.

Remarquons enfin que contrairement au cas des graphes suffixe-reconnaissables, d'ordre 1 ou supérieur, ou des graphes arbres-automatiques, nous avons considéré ici les restrictions à des ensembles de sommets accessibles dans le système de réécriture et non plus à des ensembles reconnaissables quelconques de sommets. Cela est dû au fait que l'on souhaite conserver la décidabilité du prédicat d'accessibilité dans ces graphes et que ce prédicat est défini en fonction du système de réécriture dans la preuve. Si on se restreint à un ensemble non accessible de sommets, on pourra éventuellement relier deux sommets dont l'un est accessibles à partir de l'autre dans le système de réécriture, mais pas dans le graphe considéré. Dans [Löd02] Löding a montré l'exemple d'un graphe de réécriture suffixe d'arbres dont la restriction à un ensemble de sommets reconnaissable qui n'est pas son ensemble d'accessibilité ne possède pas un $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable.

Tout comme les systèmes de réécriture de piles, les systèmes de réécriture d'arbres peuvent être utilisés pour reconnaître des langages, en considérant les étiquettes des calculs entre deux ensembles finis ou reconnaissables de configurations. La classe de langages reconnus par des systèmes de réécriture d'arbres n'est pas aussi bien connue que les langages algébriques. À notre connaissance, elle ne dispose pas d'autre description que celle d'être reconnue par un tel système. Cependant, dans sa thèse, Löding donne quelques propriétés de cette classe de langages :

Proposition 3.20 ([Löd03], 3.53, 3.54, 3.55)**.** *La classe $\mathcal{L}_{\text{GTRS}}$ des langages reconnus par les systèmes de réécriture d'arbres contient strictement les langages algébriques.*

$\mathcal{L}_{\text{GTRS}}$ est strictement incluse dans les langages contextuels.

$\mathcal{L}_{\text{GTRS}}$ est close par union.

$\mathcal{L}_{\text{GTRS}}$ n'est pas close par complémentaire, ni par intersection avec des langages reconnaissables.

Les langages de séparation que Löding utilise sont les langages $\mathcal{L}_{abc} = \{a^n b^n c^n \mid n \geq 0\}$, qui est un langage contextuel (en fait, il est même indexé d'ordre 2), et qui n'est pas reconnaissable par un système de réécriture suffixe d'arbres, et $\mathcal{L}' = \{u \mid |u|_a = |u|_b = |u|_c\}$ des mots contenant autant d'occurrences de a, de b et de c, qui est bien reconnu par un système de réécriture d'arbres, mais qui n'est pas algébrique. En réalité, $\mathcal{L}'$ n'est même pas dans la hiérarchie IO. Ceci nous permet d'affirmer que les classes de langages reconnues par les automates à pile d'ordre supérieur et les systèmes de réécriture suffixes d'arbres sont orthogonales, ce qui est donc également le cas des graphes induits par ces systèmes.

Récapitulons les propriétés que nous avons présentées sur les graphes de systèmes de réécriture suffixe d'arbres, en insistant sur les réponses aux problèmes présentés au début de ce chapitre :

- Ils disposent des représentations présentées au théorème précédent.
- Ils possèdent une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable.
- L'ensemble des sommets accessibles à partir d'un sommet donné est isomorphe à un ensemble rationnel d'arbres, et est donc calculable.
- Cette classe de graphe est close par union et par FO-interprétation.
- L'arbre binaire complet est un générateur de cette classe par interprétation à ensembles finis.

3.3.8 Hiérarchie arbre-automatiques

L'une des représentations des graphes arbres-automatiques est qu'ils sont obtenus par une interprétation à ensembles finis à partir de l'arbre binaire complet [CL07]. On peut ainsi faire un parallèle entre ces graphes et les graphes suffixe-reconnaissables d'ordre 1 qui sont obtenus par une MSO-interprétation à partir de ce même arbre binaire complet. Nous avons également vus que les graphes suffixe-reconnaissables d'ordre n sont obtenus par une MSO-interprétation précédée de $n-1$ opérations de structure arborescente de l'arbre binaire complet (ou de manière équivalente par une MSO-interprétation de $\Delta^n_{\{a,b\}}$). Dans [CL07], Colcombet et Löding définissent en s'inspirant de ces considérations la hiérarchie des graphes arbres-automatiques :

Définition 3.11 ([CL07])**.** *Un graphe $\mathcal{G}$ est arbre-automatique d'ordre n s'il est obtenu par une interprétation à ensemble fini précédées de $n-1$ structures arborescentes à partir de l'arbre binaire complet.*

Cette définition implique directement la propriété suivante :

Proposition 3.21 ([CL07])**.** *Un graphe arbre-automatique d'ordre n possède une* FO*-théorie décidable.*

De même, puisque que la hiérarchie à pile est stricte, la hiérarchie arbre-automatique l'est également.

Ces graphes n'apparaissent à notre connaissance que dans [CL07], et ne possèdent aucune représentation interne. De même, les langages qu'ils reconnaissent sont inconnus. Il est cependant possible d'y adapter le théorème 29 de [Col04] :

Théorème 3.22. *Les classes de graphes suivantes sont équivalentes :*

- *Les graphes obtenus par une interprétation à ensembles finis du dépliage d'un graphe suffixe-reconnaissable d'ordre n.*
- *Les solutions des VRS-systèmes d'équations représentés par un graphe suffixe-reconnaissable d'ordre n.*

3.4 Récaputilatif

Dans ce chapitre, nous avons décrit les classes de graphes suffixe-reconnaissables, suffixe-reconnaissables d'ordre supérieur, les graphes associés à des systèmes de réécriture suffixes d'arbres et à des transducteurs suffixes d'arbres, les graphes arbre-automatiques et la hiérarchie arbre-automatique. Les propriétés de ces graphes sont résumés à la figure 3.14, en mettant les liens principaux entre les classes, les logiques décidables et les langages reconnus. Sur cette figure, on utilise les raccourcis suivant pour qu'elle soit plus lisible :

- SR_n désigne la classe des graphes suffixe-reconnaissables d'ordre n.
- GTR désigne la classe des graphes des systèmes de réécriture suffixes d'arbres.
- GTT désigne la classe des graphes des transducteurs suffixes d'arbres.
- TAut_n désigne la classe des graphes arbre-automatiques d'ordre n.
- MSI désigne une MSO-interprétation.
- Unf désigne une opération de structure arborescente.
- FSI désigne une interprétation à ensembles finis.
- $\star$ désigne la clôture transitive.

On observe qu'il n'y a pas d'équivalent des graphes de systèmes de réécriture suffixe d'arbres et de transducteurs suffixes d'arbres aux ordres supérieurs à 1. Le chapitre suivant présente la description d'une telle classe de graphes.

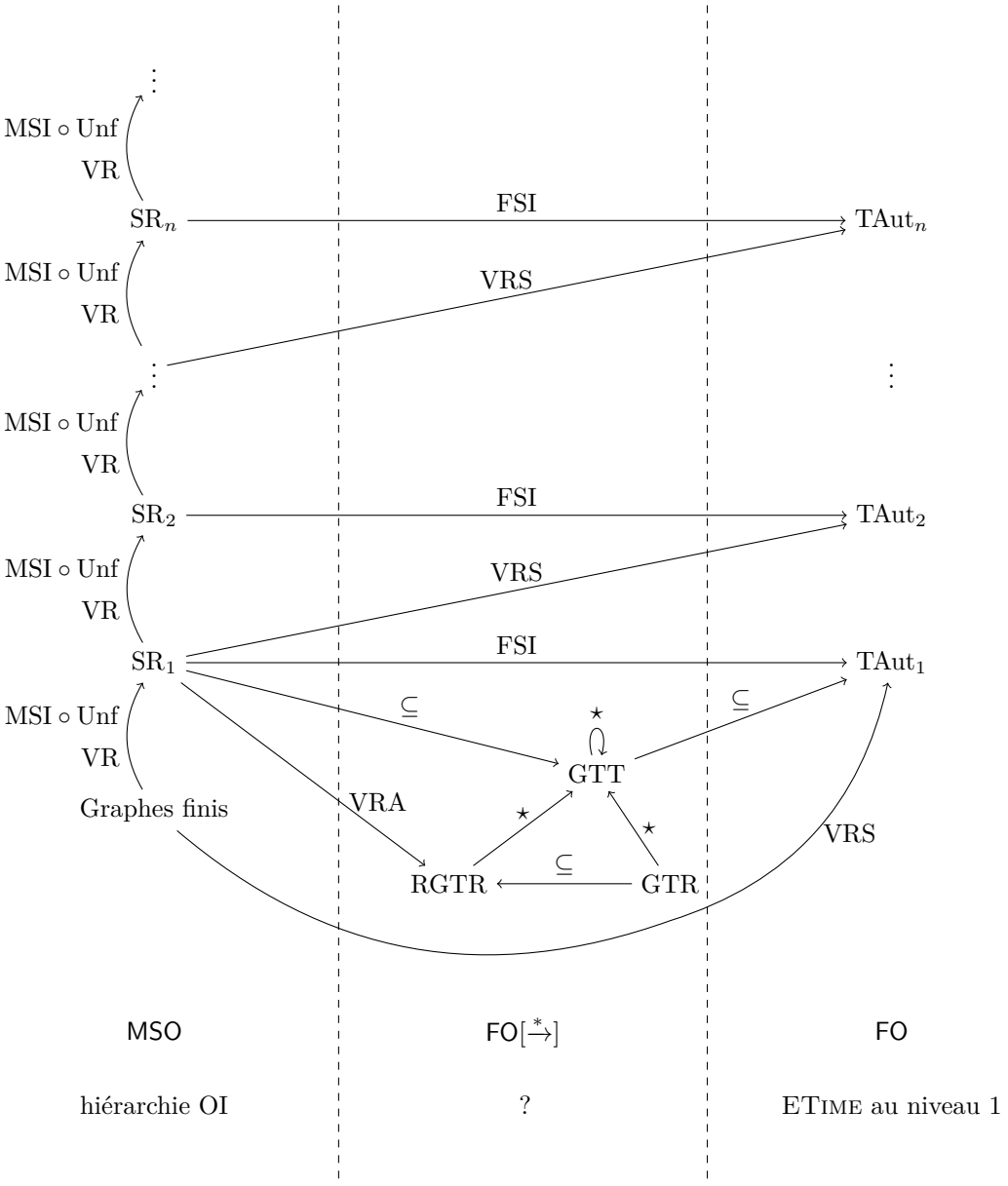


FIGURE 3.14 – Récapitulatif des résultats du chapitre.

Chapitre 4

Systèmes de réécriture suffixe d'arbres de piles

Comme l'annonçait le chapitre précédent, aucun équivalent des systèmes de réécriture suffixes d'arbres et des transducteurs suffixes d'arbres n'est connu pour les ordres supérieurs à un. Le présent chapitre propose un tel modèle : les systèmes de réécriture d'arbres de piles. Un arbre de pile d'ordre n est simplement un arbre dont les nœuds sont étiquetés par des piles d'ordre $n-1$. Les résultats de ce chapitre ont été présentés à CSR 2015 [Pen15]. Nous commençons par présenter le modèle des arbres de piles et les opérations que nous autorisons pour définir les systèmes de réécriture d'arbres de piles. Nous montrons ensuite que ce modèle étend à la fois les systèmes de réécriture de pile et les systèmes de réécriture suffixe d'arbres. Nous définissons ensuite une notion de reconnaissabilité sur les opérations, ce qui permet de dériver une notion de reconnaissabilité sur les arbres de piles, de la même manière que dans le cas des piles d'ordre supérieurs. Puis nous utilisons ces résultats pour définir des classes de graphes associés aux systèmes de réécriture suffixe d'arbres de piles et aux automates d'opérations qui étendent la notion de graphe de réécriture suffixe d'arbres et de graphes de transducteurs suffixes d'arbres, et nous montrons qu'ils possèdent une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable. Enfin, nous abordons la question des langages reconnus par ces graphes, en présentant un exemple d'un tel langage, même si la classe exacte des langages reconnus par ces graphes nous est encore inconnue.

4.1 Définition du modèle

4.1.1 Arbres de piles

On commence par définir formellement la notion d'arbre de piles. Dans tout ce chapitre, Γ désignera un alphabet fini.

Définition 4.1. *Un* arbre de piles *d'ordre n, ou* n-arbre de piles *sur l'alphabet Γ, est un arbre étiqueté par $Stacks_{n-1}(\Gamma)$. On introduit l'ensemble $ST_n(\Gamma) = \mathcal{T}(Stacks_{n-1}(\Gamma))$ des n-arbres de piles (ou plus simplement ST_n s'il n'y a pas d'ambiguïté sur Γ).*

Remarquons qu'un n-arbre de piles de degré 1 est isomorphe à une n-pile, et que $ST_1(\Gamma) = \mathcal{T}(\Gamma)$. La figure 4.1 montre l'exemple d'un 3-arbre de piles. La notion d'arbres de piles étend ainsi à la fois les notions de piles d'ordres supérieurs et d'arbres.

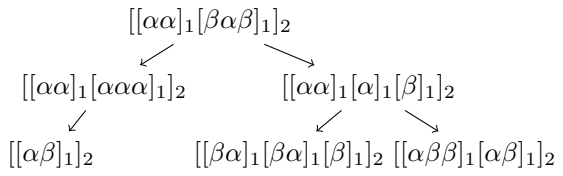


FIGURE 4.1 – Un 3-arbre de pile.

4.1.2 Opérations

Opérations de bases

Nous détaillons maintenant les opérations de base que l'on peut appliquer sur un n-arbre de piles. Étant donné que l'on souhaite que notre modèle étende les systèmes de réécriture de piles, notamment que sur des arbres de piles unaires on récupère exactement ce modèle, nous considèrerons naturellement l'extension des opérations des piles aux arbres de piles, qui s'appliqueront à l'une des feuilles de l'arbre, et nous aurons deux nouvelles opérations à l'ordre n nous permettant de créer et de détruire plusieurs feuilles. Nous commençons par définir la relation associée à une opération de base d'arbre de pile. Il y a en général plusieurs positions d'un arbre où une opération donnée peut être appliquée. Nous définissons donc ensuite la relation définie par *l'application localisée* d'une opération à une position donnée d'un arbre, donnée par l'indice de la feuille dans l'ordre lexicographique des feuilles.

Définition 4.2. *On considère l'ensemble des opérations de base sur ST_n, $TOps_n = \mathrm{Ops}_{n-1} \cup \{\mathrm{copy}_n^k, \overline{\mathrm{copy}}_n^k \mid 1 \leq k \leq d\}$. La relation associée à une opération est définie comme suit :*

- $\mathrm{r}_\theta = \{(t,t') \mid \mathrm{dom}(t) = \mathrm{dom}(t') \wedge \exists u \in \mathrm{fr}(t), \forall v \neq u, t(v) = t'(v) \wedge \mathrm{r}_\theta(t(u), t'(u))\}$, *pour* $\theta \in \mathrm{Ops}_{n-1}$.
- $\mathrm{r}_{\mathrm{copy}_n^k} = \{(t,t') \mid \exists u \in \mathrm{fr}(t), t' = t \cup \{uj \mapsto t(u) \mid 1 \leq j \leq k\}$.
- $\mathrm{r}_{\overline{\mathrm{copy}}_n^k} = (\mathrm{r}_{\mathrm{copy}_n^k})^{-1}$.

Définition 4.3. *Étant donnée un entier i, la relation définie par l'application localisée d'une opération de base est définie comme suit :*

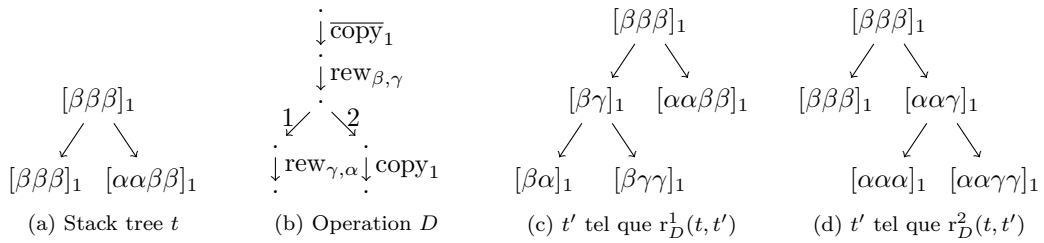
- $\mathrm{r}_\theta^i = \{(t,t') \mid \mathrm{dom}(t) = \mathrm{dom}(t') \wedge \forall v \neq u_i, t(v) = t'(v) \wedge \mathrm{r}_\theta(t(u_i), t'(u_i))\}$, *pour* $\theta \in \mathrm{Ops}_{n-1}$.
- $\mathrm{r}^i_{\mathrm{copy}_n^k} = \{(t,t') \mid t' = t \cup \{u_i j \mapsto t(u_i) \mid 1 \leq j \leq k\}$.
- $\mathrm{r}^i_{\overline{\mathrm{copy}}_n^k} = (\mathrm{r}^i_{\mathrm{copy}_n^k})^{-1}$.

Dans chacune des égalités précédentes, on considère que $\mathrm{fr}(t) = \{u_1, \cdots, u_{|\mathrm{fr}(t)|}\}$, et sont ordonnées dans cet ordre suivant l'ordre lexicographique sur $\{1,2\}^*$. Si $i > |\mathrm{fr}(t)|$, alors il n'existe aucun arbre de piles t' tel que $\mathrm{r}_\theta^i(t,t')$.

Par soucis de simplicité, tout comme au chapitre précédent, nous considérons uniquement le cas où les arbres de piles ont une arité d'au plus 2. Ceci dit, tous les résultats que nous présentons s'étendent facilement au cas général.

Insuffisance de la composition

La définition précédente contient toutes les opérations de bases sur des piles. Si on se restreint aux arbres de piles unaires, on peut facilement observer que les opérations de $TOps_n$ sont exactement celles de Ops_n en remarquant que copy_n^1 définit la même relation que copy_n. Dans ce cas, la composition de relations d'opérations de base définit toutes les relations définies par des opérations de Ops_n^*. Comme annoncé auparavant, ST_1 est l'ensemble des arbres étiquetés par Γ,



(a) Stack tree t (b) Operation D (c) t' tel que $\text{r}^1_D(t,t')$ (d) t' tel que $\text{r}^2_D(t,t')$

FIGURE 4.2 – L'application d'une opération D à un arbre de piles t.

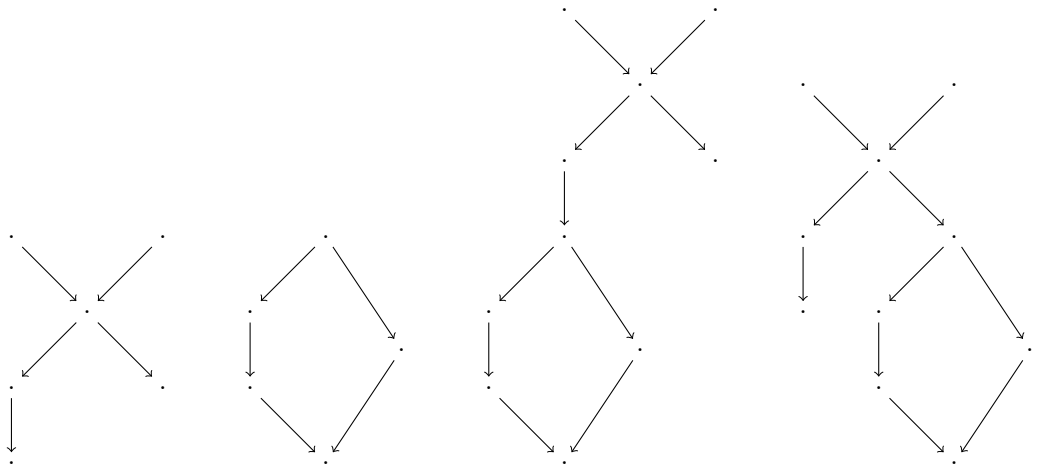
et on souhaite également étendre toutes les relations définies par des règles de réécriture suffixe d'arbres. Cependant, la composition des relations définies par des opérations de bases n'est pas suffisante pour obtenir ces relations. En effet, une règle de réécriture d'arbre suffixe close (g, d) exprime le remplacement d'un sous-arbre clos de taille arbitraire g d'un arbre $s = c[g]$ par d, donnant l'arbre $c[d]$. Composer des relations définies par des opérations d'arbres de piles de base ne permet pas d'exprimer de telles opérations, car il n'y a aucune garantie que deux opérations successives seront appliquées à la même partie de l'arbre. Nous devons donc considérer une façon de composer des opérations de bases qui assure qu'elle agiront sur un seul sous-arbre. Avec nos notations, l'effet d'une règle d'arbre close suffixe peut être vue comme l'application *localisée* d'une suite de rew et de $\overline{\text{copy}}_1^d$ suivie par une suite de rew et de copy_1^d. Nous adopterons une présentation des *opérations composées* comme des DAGs, qui nous permettrons de spécifier les positions relatives de l'applications des opérations de bases les composant. Toutefois tout DAG ne représentera pas une opération composée valide. Nous définirons donc tout d'abord la classe de DAG appropriée, ainsi qu'une opération de concaténation sur cette classe. Un exemple du modèle que nous voulons définir est présenté en figure 4.2.

Concaténation de DAGs.

Pour définir formellement notre notion d'opération composée, on a besoin d'une notion technique de concaténation de DAGs, qui représentera une *composition localisée* des opérations. On considère des DAGs dont les sommets sont ordonnés totalement, de manière à ce que l'ordre respecte les arcs (c'est-à-dire que si x est un ancêtre de y, alors $x \leq y$). Étant donnés deux DAGs D et D' tels que l'ensemble O_D des sommets sortants (c'est-à-dire sans arc sortant) de D soit $O_D = \{b_1, \ldots, b_\ell\}$ (ordonnés dans cet ordre) et l'ensemble $I_{D'}$ des sommets entrants (c'est-à-dire sans arc entrant) de D' soit $I_{D'} = \{a'_1, \ldots, a'_{k'}\}$ (ordonnés dans cet ordre), et deux indices i et j tels que $1 \leq i \leq \ell$ et $1 \leq j \leq k'$, on note $D \cdot_{i,j} D'$ l'unique DAG D'' obtenu en identifiant le $(i+m)^{\text{ème}}$ nœud sortant de D avec le $(j+m)^{\text{ème}}$ nœud d'entrée de D' pour tout $m \geq 0$ tel que b_{i+m} et a'_{j+m} existent tous les deux. Formellement, en notant $d = \min(\ell - i, k' - j) + 1$ le nombre de sommets identifiés, on a $D'' = \text{merge}_f(D \uplus D')$ où $\text{merge}_f(D)$ est le DAG dont l'ensemble de sommets est $f(V_D)$ est l'ensemble d'arcs est $\{(f(x), \gamma, f(x')) \mid (x, \gamma, x') \in E_D\}$, et $f(x) = b_{i+m}$ si $x = a'_{j+m}$ pour $0 \leq m \leq d$, et $f(x) = x$ dans les autres cas. On appelle D'' la (i,j)-concaténation de D et D'. Remarquons que la (i,j)-concaténation de deux DAGs connexes est connexe.

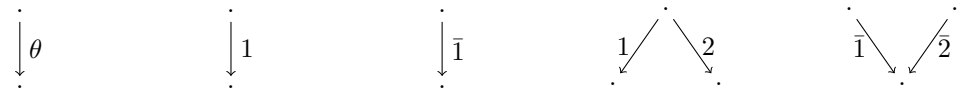
Opérations Composées

Nous représentons les opérations composées par des DAGs. Nous définissons d'abord l'ensemble $\mathcal{D}_n = \{D_\theta \mid \theta \in TOps_n\}$ des DAGs représentant les opérations de bases représentés sur la figure 4.4. On définit récursivement les opération composées comme représenté sur la figure 4.5.



(a) Un DAG D_1 (b) Un DAG D_2 (c) Le DAG $D_1 \cdot_{1,1} D_2$ (d) Le DAG $D_1 \cdot_{2,1} D_2$

FIGURE 4.3 – Exemple de la composition de deux DAGs. Les sommets sont ordonnés de gauche à droite, et suivant les arcs.



(a) D_θ (b) $D_{\mathrm{copy}^1_n}$ (c) $D_{\overline{\mathrm{copy}}^1_n}$ (d) $D_{\mathrm{copy}^2_n}$ (e) $D_{\overline{\mathrm{copy}}^2_n}$

FIGURE 4.4 – DAGs des opérations de bases sur les n-arbres de piles (ici θ appartient à Ops_{n-1}).

Définition 4.4. *Un DAG D est une* opération composée *(ou plus simplement une* opération*) si l'une des conditions suivantes est satisfaite :*

1. $D = \square$;
2. $D = (D_1 \cdot_{1,1} D_\theta) \cdot_{1,1} D_2$, *avec* $|O_{D_1}| = |I_{D_2}| = 1$ *et* $\theta \in \mathrm{Ops}_{n-1} \cup \{\mathrm{copy}^1_n, \overline{\mathrm{copy}}^1_n\}$;
3. $D = ((D_1 \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} D_3) \cdot_{1,1} D_2$, *avec* $|O_{D_1}| = |I_{D_2}| = |I_{D_3}| = 1$;
4. $D = (D_1 \cdot_{1,1} (D_2 \cdot_{1,2} D_{\overline{\mathrm{copy}}^2_n})) \cdot_{1,1} D_3$ *avec* $|O_{D_1}| = |O_{D_2}| = |I_{D_3}| = 1$;
5. $D = ((((D_1 \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} D_3) \cdot_{1,1} D_2) \cdot_{1,1} D_{\overline{\mathrm{copy}}^2_n}) \cdot_{1,1} D_4$, *avec* $|O_{D_1}| = |I_{D_2}| = |O_{D_2}| = |I_{D_3}| = |O_{D_3}| = |I_{D_4}| = 1$;

où D_1, D_2, D_3 et D_4 sont des opérations composées.

Par ailleurs, les sommets de D sont ordonnés récursivement de telle manière que tous les sommets de D_i soient plus petits que ceux de D_{i+1} dans la définition ci-dessus, l'ordre sur $\square$ étant l'ordre trivial. Cela induit en particulier un ordre sur les sommets entrants et sortants de D.

Définition 4.5. *Étant donnée une opération composée D, on définit r^i_D, la relation associée à son* application localisée *commençant à la $i^{\text{ème}}$ feuille d'un arbre de piles t, comme suit :*

1. *Si $D = \square$, alors $\mathrm{r}^i_D = \mathrm{id}$.*
2. *Si $D = (D_1 \cdot_{1,1} D_\theta) \cdot_{1,1} D_2$ avec $\theta \in \mathrm{Ops}_{n-1} \cup \{\mathrm{copy}^1_n, \overline{\mathrm{copy}}^1_n\}$, alors $\mathrm{r}^i_D = \mathrm{r}^i_{D_2} \circ \mathrm{r}^i_\theta \circ \mathrm{r}^i_{D_1}$.*
3. *Si $D = ((D_1 \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} D_3) \cdot_{1,1} D_2$, alors $\mathrm{r}^i_D = \mathrm{r}^i_{D_2} \circ \mathrm{r}^{i+1}_{D_3} \circ \mathrm{r}^i_{\mathrm{copy}^2_n} \circ \mathrm{r}^i_{D_1}$.*

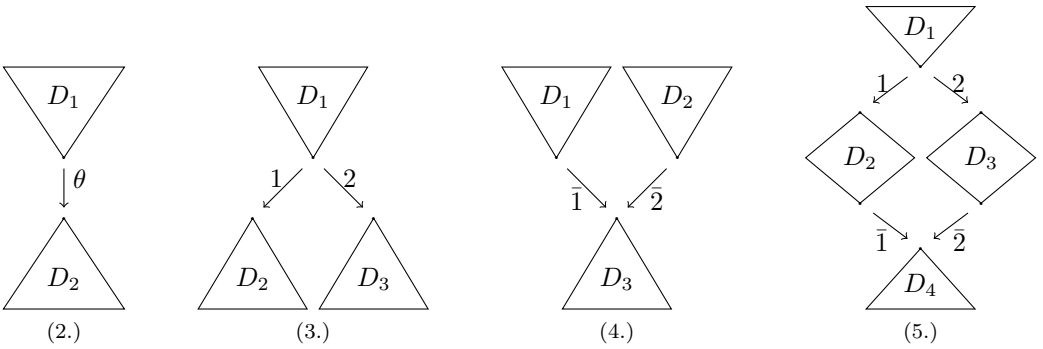


FIGURE 4.5 – Décompositions possibles des opérations composées, la numérotation correspondant aux cas de la définition 4.4.

4. *Si* $D = (D_1 \cdot_{1,1} (D_2 \cdot_{2,1} D_{\overline{\mathrm{copy}_n^2}})) \cdot_{1,1} D_3$, *alors* $\mathrm{r}_D^i = \mathrm{r}_{D_3}^i \circ \mathrm{r}_{\overline{\mathrm{copy}_n^2}}^i \circ \mathrm{r}_{D_2}^{i+1} \circ \mathrm{r}_{D_1}^i$.
5. *Si* $D = ((((D_1 \cdot_{1,1} D_{\mathrm{copy}_n^2}) \cdot_{2,1} D_3) \cdot_{1,1} D_2) \cdot_{1,1} D_{\overline{\mathrm{copy}_n^2}}) \cdot_{1,1} D_4$,
alors $\mathrm{r}_D^i = \mathrm{r}_{D_4}^i \circ \mathrm{r}_{\overline{\mathrm{copy}_n^2}}^i \circ \mathrm{r}_{D_2}^i \circ \mathrm{r}_{D_3}^{i+1} \circ \mathrm{r}_{\mathrm{copy}_n^2}^i \circ \mathrm{r}_{D_1}^i$.

La relation définie par D *est* $\mathrm{r}_D = \bigcup_i \mathrm{r}_D^i$, *c'est à dire l'union de toutes ses applications localisées.*

Remarque 4.1. *Une opération admet éventuellement plusieurs décompositions dans la définition 4.4. Toutefois, son application est bien définie, la décomposition étant localement confluente. Pour l'observer, il suffit d'observer tous les cas possible et de remarquer que quelque soit l'ordre dans lequel on décompose l'opération, on obtient la même relation. Cette observation ne présentant aucune autre difficulté qu'être fastidieuse, nous la laissons au lecteur.*

La figure 4.2 montre l'exemple de l'application d'une opération composée. Étant donné une opération D, on définit $\overline{D}$ l'opération obtenue en inversant tous les arcs de D et en inversant les sommets entrants et sortants. Plus formellement, pour $\theta \in \mathrm{Ops}_{n-1}(\Gamma)$, on définit $\overline{D_\theta} = D_{\overline{\theta}}$, $\overline{D_{\mathrm{copy}_n^d}} = D_{\overline{\mathrm{copy}}_n^d}$, $\overline{D_{\overline{\mathrm{copy}}_n^d}} = D_{\mathrm{copy}_n^d}$, et pour toutes opérations D_1, D_2, $\overline{D_1 \cdot_{i,j} D_2} = \overline{D_2} \cdot_{j,i} \overline{D_1}$. On a pour toute opération D, $\mathrm{r}_{\overline{D}} = \mathrm{r}_D^{-1}$. Finalement, étant donné un k-uplet d'opérations $\vec{D} = (D_1, \ldots, D_k)$ de degrés entrants respectifs $d_1, \ldots, d_k$ et un k-uplet d'indices $\mathbf{i} = (i_1, \ldots, i_k)$ avec $i_{j+1} \geq i_j + d_j$ pour tous $1 \leq j < k$, on note par $\mathrm{r}_{\vec{D}}^{\mathbf{i}} = \mathrm{r}_{D_1}^{i_1} \circ \cdots \circ \mathrm{r}_{D_k}^{i_k}$ la relation définie par l'application parallèle de chaque D_j à la position i_j, et $\mathrm{r}_{\vec{D}}$ l'union de toutes les relations ainsi définies.

Puisque la (i,j)-concaténation de deux opérations comme définie ci-dessus n'est pas forcément une opération, nous devons nous réduire aux résultats bien formés selon la définition 4.4. Étant donnés D et D', on note $D \cdot D' = \{D \cdot_{i,j} D' \mid D \cdot_{i,j} D' \text{ est un opération}\}$. Étant donné $n > 1$, on définit [1] $D^n = \bigcup_{i<n} D^i \cdot D^{n-i}$, et on note $D^* = \bigcup_{n\geq 0} D^n$ l'ensemble des *itérations* de D. Ces notations s'étendent naturellement aux ensembles d'opérations.

Proposition 4.1. $\mathcal{D}_n^*$ *est exactement l'ensemble des opérations bien formées.*

Démonstration. Rappelons que $\mathcal{D}_n$ est l'ensemble des DAGs associés aux opérations de base. Par définition, tout DAG dans $\mathcal{D}_n^*$ est une opération. Inversement, d'après la définition 4.4, toute opération peut être décomposée comme une concaténation d'opérations de $\mathcal{D}_n$. □

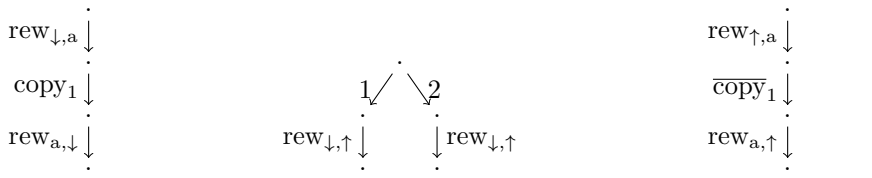
1. Cette définition peu habituelle est nécessaire car $\cdot$ n'est pas associative. Par exemple, $(D_{\mathrm{copy}_n^2} \cdot_{2,1} D_{\mathrm{copy}_n^2}) \cdot_{1,1} D_{\mathrm{copy}_n^2}$ est dans $(D_{\mathrm{copy}_n^2})^2 \cdot D_{\mathrm{copy}_n^2}$ mais pas dans $D_{\mathrm{copy}_n^2} \cdot (D_{\mathrm{copy}_n^2})^2$.

Systèmes de réécriture suffixe d'arbres de piles

Avec les définitions précédentes, on définit les systèmes de réécriture suffixe d'arbres de piles de manière analogue aux systèmes de réécriture de pile et aux systèmes de réécriture suffixe d'arbres.

Définition 4.6. *Un système de réécriture suffixe de n-arbres de piles R étiqueté par Σ est la donnée de $|\Sigma|$ ensembles finis R_a d'opérations. Il existe éventuellement un ensemble R_ε d'opérations silencieuses étiquetées par $\varepsilon \notin \Sigma$. Il induit le graphe $\mathcal{G}_R = (ST_n, E)$ avec $E = \{(t, a, t') \mid \exists D \in R_a, \mathrm{r}_D(t, t')\}$.*

Exemple 4.1. *On fixe un alphabet Σ et deux symboles spéciaux $\uparrow$ et $\downarrow$. on considère $ST_2(\Sigma \cup \{\uparrow, \downarrow\})$. On définit le système de réécriture R, avec pour tout $a \in \Sigma$, $R_a = \{D_a\}$ et $R_\varepsilon = \{\mathrm{Dupl}\} \cup \{P_a \mid a \in \Sigma\}$. Ces règles sont données à la figure 4.6. On représente à la figure 4.7 la restriction du graphe associé à ce système aux sommets accessibles depuis l'arbre de pile contenant un seul nœud étiqueté par $[\downarrow]_1$. Pour des raisons de lisibilité, nous ne représentons pas les étiquettes des nœuds, aussi pour faciliter leur identification, nous étiquetons les arêtes par les opérations et non par les lettres étiquetant ces opérations.*



(a) P_a, pour chaque $a \in \Sigma$ (b) *Dupl* (c) D_a, for every $a \in \Sigma$

FIGURE 4.6 – Les règles du système de réécriture de l'exemple 4.1.

4.2 Extensions des piles d'ordre supérieur et des arbres

Dans ce paragraphe, on s'intéresse au lien entre les systèmes de réécriture d'arbres de pile, les systèmes de réécriture d'arbre et les systèmes de réécriture de piles. On va montrer que les systèmes de réécriture d'arbres de piles sont des généralisations des deux autres en montrant qu'à l'ordre 1, les systèmes de réécritures d'arbres de piles sont exactement les systèmes de réécriture d'arbres et que les systèmes de réécriture d'arbres de piles unaires sont exactement les systèmes de réécriture de piles.

Proposition 4.2. *Les notions de systèmes de réécriture suffixe d'arbres et de systèmes de réécriture suffixe de 1-arbres de piles coïncident.*

Démonstration. On a déjà vu que ST_1 est directement l'ensemble des arbres étiquetés par Γ.

Étant donné un arbre t, on définit son DAG constructif associé D_t par induction comme suit :

- Si $t = \{\varepsilon \to a\}$, $D_t = D_{\mathrm{rew}_{a_0,a}}$, où a_0 est une lettre de Γ donnée.
- Si $t = \{\varepsilon \to a\} \cup 1 \cdot t_1 \cup 2 \cdot t_2$, $D_t = D_{\mathrm{rew}_{a_0,a}} \cdot_{1,1} ((D_{\mathrm{copy}_1^2} \cdot_{2,1} (D_{\mathrm{rew}_{a,a_0}} \cdot_{1,1} D_{t_2}) \cdot_{1,1} (D_{\mathrm{rew}_{a,a_0}} \cdot_{1,1} D_{t_1}))$.

où $d.t$ désigne la fonction définie par $d.t(u) = t(d.u)$ pour tout u.

Lemme 4.3. *Étant donnés t, t' et t_r trois arbres, on a $\mathrm{r}_{D_{t_r}}(t, t')$ si et seulement il existe C un arbre et u une feuille de C tels que si $t = C[a_0]_u$ et $t' = C[t_r]_u$.*

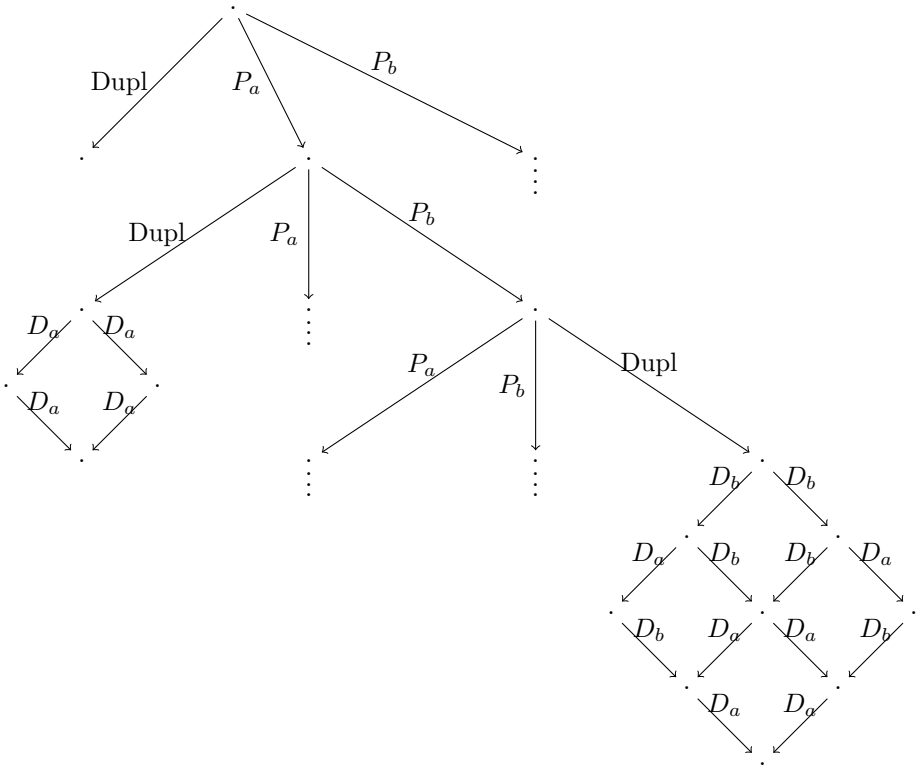


FIGURE 4.7 – Le graphe du système de réécriture suffixe d'arbres de piles restreint aux sommets accessibles depuis l'arbre à un seul nœud étiqueté par $[\downarrow]_1$

Démonstration. Par induction sur t_r, on obtient ce lemme.

Si $t_r = \{\varepsilon \to a\}$, on a $D_{t_r} = D_{\mathrm{rew}_{\mathrm{a}_0,\mathrm{a}}}$. Si $\mathrm{r}_{D_{t_r}}(t,t')$ est vrai, il existe, par définition un $i \geq |\mathrm{fr}(t)|$ tel que $\mathrm{r}^i_{\mathrm{rew}_{\mathrm{a}_0,\mathrm{a}}}(t,t')$. On a donc, pour tout $v \neq u_i$, $t'(v) = t(v)$ et $\mathrm{r}_{\mathrm{rew}_{\mathrm{a}_0,\mathrm{a}}}(t(u_i), t'(u_i))$, et ainsi, il existe bien un arbre C tel que $t = C[a_0]_{u_i}$ et $t' = C[a]_{u_i} = C[t_r]_{u_i}$, puisque autrement D_{t_r} n'est pas applicable à t.

Si $t_r = \{\varepsilon \to a\} \cup 1 \cdot t_1 \cup 2 \cdot t_2$, on a $D_{t_r} = D_{\mathrm{rew}_{\mathrm{a}_0,\mathrm{a}}} \cdot_{1,1} ((D_{\mathrm{copy}^2_1} \cdot_{2,1} (D_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}} \cdot_{1,1} D_{t_2}) \cdot_{1,1} (D_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}} \cdot_{1,1} D_{t_1}))$. Si $\mathrm{r}_{D_{t_r}}(t,t')$ est vrai, il existe par définition un $i \geq |\mathrm{fr}(t)|$ tel que $\mathrm{r}^i_{D_{t_r}}(t,t')$, donc si $\mathrm{r}^i_{D_{t_1}} \circ \mathrm{r}^i_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}} \circ \mathrm{r}^{i+1}_{D_{t_2}} \circ \mathrm{r}^{i+1}_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}} \circ \mathrm{r}^i_{\mathrm{copy}^2_1} \circ \mathrm{r}^i_{\mathrm{rew}_{\mathrm{a}_0,\mathrm{a}}}(t,t')$.

De même que précédemment, on a $\mathrm{r}^i_{D_{t_1}} \circ \mathrm{r}^i_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}} \circ \mathrm{r}^{i+1}_{D_{t_2}} \circ \mathrm{r}^{i+1}_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}} \circ \mathrm{r}^i_{\mathrm{copy}^2_1}(C[a]_{u_i}, t')$, et $t = C[a_0]_{u_i}$, avec C un arbre.

Par définition de copy^2_1, on a $\mathrm{r}^i_{D_{t_1}} \circ \mathrm{r}^i_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}} \circ \mathrm{r}^{i+1}_{D_{t_2}} \circ \mathrm{r}^{i+1}_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}}(r_1, t')$, avec $r_1 = \{\varepsilon \to a, 1 \to a, 2 \to a\}$.

Par définition de $\mathrm{rew}_{\mathrm{a},\mathrm{b}}$, on a $\mathrm{r}^i_{D_{t_1}} \circ \mathrm{r}^i_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}} \circ \mathrm{r}^{i+1}_{D_{t_2}}(r_2, t')$, avec $r_2 = \{\varepsilon \to a, 1 \to a, 2 \to a_0\}$.

Par hypothèse de récurrence, on a $\mathrm{r}^i_{D_{t_1}} \circ \mathrm{r}^i_{\mathrm{rew}_{\mathrm{a},\mathrm{a}_0}}(r_2, t')$, avec$r_3 = \{\varepsilon \to a, 1 \to a\} \cup 2 \cdot t_2$.

De même que les deux étapes précédentes, on obtient $t' = C[r_4]_{u_i}$, avec $r_4 = \{\varepsilon \to a\} \cup 1 \cdot t_1 \cup 2 \cdot t_2 = t_r$. □

Étant donnée une règle de GTRS (t_1, t_2), on lui associe le DAG $D_{t_1,t_2} = \overline{D_{t_1}} \cdot_{1,1} D_{t_2}$.

Par le lemme précédent, on a $\mathrm{r}_{D_{t_1,t_2}}(t,t')$ si et seulement si $t = C[t_1]_u$ et $t' = C[t_2]_u$, ce qui est par définition si et seulement si $\mathrm{r}_{(t_1,t_2)}(t,t')$.

Étant donné un DAG D ne contenant pas de $D_{\overline{\mathrm{copy}^d_1}}$, et une lettre $a \in \Gamma$, on définit par induction $t_{D,a}$ l'arbre associé :

- Si $D = \square$, $t_{D,a} = \{\varepsilon \to a\}$.
- Si $D = D_{\mathrm{rew}_{\mathrm{c},\mathrm{b}}}$, $t_{D,a} = \{\varepsilon \to b\}$ si $c = a$ et n'est pas défini sinon.
- Si $D = D_1 \cdot_{1,1} ((D_{\mathrm{copy}^2_1} \cdot_{2,1} D_3) \cdot_{1,1} D_2)$, on a :
  - Si $D_1 = \square$, $t_{D,a} = \{\varepsilon \to a, 1 \to t_{D_2,a}, 2 \to t_{D_3,a}\}$.
  - Si $D_1 = D_{\mathrm{rew}_{\mathrm{a},\mathrm{c}}} \cdot_{1,1} D'_1 \cdot_{1,1} D_{\mathrm{rew}_{\mathrm{d},\mathrm{b}}}$ ou $D_1 = D_{\mathrm{rew}_{\mathrm{a},\mathrm{b}}}$, $t_{D,a} = \{\varepsilon \to b, 1 \to t_{D_2,b}, 2 \to t_{D_3,b}\}$

Lemme 4.4. *Pour tous D DAG sans $D_{\overline{\mathrm{copy}^d_1}}$ et t_1, t_2 arbres, on a $\mathrm{r}_D(t_1,t_2)$ si et seulement si $t_1 = C[a]_u$ et $t_2 = C[t_{D,a}]_u$, où a est une lettre de Γ.*

Démonstration. On le prouve par induction sur D :

Si $D = \mathrm{rew}_{\mathrm{a},\mathrm{b}}$, on a $t_{D,a} = \{\varepsilon \to b\}$, par définition. On a également par définition $\mathrm{r}_D(t_1,t_2)$ si et seulement si il existe un arbre C et $u \in \mathrm{fr}(C)$ tel que $t_1 = C[a]_u$ et $t_2 = C[b]_u = C[t_{D,a}]_u$.

Si $D = D_1 \cdot_{1,1} ((D_{\mathrm{copy}^2_1} \cdot_{2,1} D_3) \cdot_{1,1} D_2)$, on a $\mathrm{r}_D(t_1,t_2)$ si et seulement si il existe $i \leq \mathrm{fr}(t_1)$ tel que $\mathrm{r}^i_D(t_1,t_2)$ et donc, $\mathrm{r}^i_{D_2} \circ \mathrm{r}^{i+1}_{D_3} \circ \mathrm{r}^i_{\mathrm{copy}^2_1} \circ \mathrm{r}^i_{D_1}(t_1,t_2)$. Étant donné que D ne contient pas de $\overline{\mathrm{copy}^d_1}$, on a nécessairement que D_1 est une suite de $\mathrm{rew}_{\mathrm{a},\mathrm{b}}$ ou $\square$. On a donc $\mathrm{r}^i_{D_2} \circ \mathrm{r}^{i+1}_{D_3} \circ \mathrm{r}^i_{\mathrm{copy}^2_1}(r_1,t_2)$, avec $r_1 = t_1$ si $D_1 = \square$ et $r_1 = t_1[b]_{u_i}$ si D_1 est une suite de $\mathrm{rew}_{\mathrm{a},\mathrm{b}}$ terminant par un b.

Par définition de copy^2_1, on a $\mathrm{r}^i_{D_2} \circ \mathrm{r}^{i+1}_{D_3}(r_2,t_2))$, avec $r_2 = t_1[s_1]_{u_i}$ où $s_1 = \{\varepsilon \to r_1(u_i), 1 \to r_1(u_i), 2 \to r_1(u_i)\}$.

Par hypothèse de récurrence (deux fois), on a $t_2 = t_1[s_2]_{u_i}$ avec $s_2 = \{\varepsilon \to r_1(u_i)\} \cup 1 \cdot t_{D_2,t_2(\varepsilon)} \cup \cdot t_{D_3,t_2(\varepsilon)}$, ce qui est bien $t_{D,a}$. □

On considère un DAG D. Si $D = D_1 \cdot_{1,1} ((((D_{\mathrm{copy}^2_1} \cdot_{2,1} D_3) \cdot_{1,1} D_2) \cdot_{1,1} D_{\overline{\mathrm{copy}^2_1}}) \cdot_{1,1} D_4$, alors il est équivalent à $D_1 \cdot_{1,1} D_4$ ou à $D_1 \cdot_{1,1} D_{\mathrm{rew}_{\mathrm{a},\mathrm{a}}} \cdot_{1,1} D_4$ pour $a \in \Gamma$. En effet, les parties composées de D_2 et D_3 peuvent uniquement tester l'étiquette du nœud dont elles viennent, et ce nœud est une lettre.

On considère donc un DAG $D = D_1 \cdot_{1,1} D_2$ tel que D_1 ne contienne pas de $D_{\mathrm{copy}^d_1}$ et D_2 ne contienne pas de $D_{\overline{\mathrm{copy}^d_1}}$. On lui associe l'ensemble de règles de GTRS $\tau_a = (t_{\overline{D_1},a}, t_{D_2,a})$, pour

tout $a \in \Gamma$. On a bien, pour tous arbres t, t', il existe un $a \in \Gamma$ tel que $\mathrm{r}_{\tau_a}(t, t')$ si et seulement si $\mathrm{r}_D(t, t')$.

□

Proposition 4.5. *Si on restreint les arbres de piles de STn aux arbres unaires, les systèmes de réécriture suffixe de n-arbres de piles qui peuvent s'y appliquer sont équivalents aux systèmes de réécriture de n-piles.*

Démonstration. On considère la traduction suivante des n-arbres de piles unaires vers les n-piles :

Soit t un n-arbre de piles unaire, on considère $p(t) = [t(\varepsilon), \cdots, t(u)]_n$ où u est l'unique élément de $\mathrm{fr}(t)$.

Étant donné une opération composée unaire D, on considère l'opération $o(D) \in \mathrm{Ops}_n^*$ définie comme suit :

- $o(\Box) = \mathrm{id}$,
- $o(D_\theta \cdot_{1,1} D') = \theta \cdot o(D')$, pour $\theta \in \mathrm{Ops}_{n-1}$,
- $o(D_{\mathrm{copy}_n^1} \cdot_{1,1} D') = \mathrm{copy}_n \cdot o(D')$.
- $o(D_{\overline{\mathrm{copy}}_n^1} \cdot_{1,1} D') = \overline{\mathrm{copy}}_n \cdot o(D')$.

On a par induction sur D que pour tous n-arbres unaires t, t' et tout DAG unaire D, $\mathrm{r}_D(t, t')$ si et seulement si $pr_{o(D)}(p(t), p(t'))$:

- Si $D = \Box$, $t = t'$ et $p(t') = p(t)$,
- Si $D = D' \cdot_{1,1} D_\theta$, il existe t'' tel que $\mathrm{r}_{D'}(t, t'')$ et $\mathrm{r}_\theta(t'', t')$. Par définition, on a $p(t') = [t''(\varepsilon), \cdots, t''(u_{|u|-1}), \alpha]_n$, avec $\mathrm{r}_\theta(t''(u_{|u|}), \alpha)$, on a donc bien $\mathrm{r}_\theta(p(t''), p(t'))$. Par induction, on a $\mathrm{r}_{o(D')}(p(t), p(t''))$, et donc on a bien $\mathrm{r}_{o(D'D_\theta)}(t, t')$.
- Si $D = D_{\mathrm{copy}_n^1} \cdot_{1,1} D'$, il existe t'' tel que $\mathrm{r}_{D'}(t, t'')$ et $\mathrm{r}_{\mathrm{copy}_n^1}(t'', t')$. On a donc $t' = t'' \cup \{u1 \to t''(u)\}$ et $p(t') = [t''(\varepsilon), \cdots, t''(u), t''(u)]_n$. On a donc $\mathrm{r}_{\mathrm{copy}_n}(p(t''), p(t'))$, et par induction, on a $\mathrm{r}_{o(D')}(p(t), p(t''))$. On a bien $\mathrm{r}_{o(\mathrm{copy}_n^1 \cdot D')}(p(t), p(t'))$
- Si $D = D_{\overline{\mathrm{copy}}_n^1} \cdot_{1,1} D'$, il existe t'' tel que $\mathrm{r}_{D'}(t, t'')$ et $\mathrm{r}_{\overline{\mathrm{copy}}_n^1}(t'', t')$. On a donc $t'' = t' \cup \{u1 \to t'(u)\}$, $p(t'') = [t''(\varepsilon), \cdots, t''(u), t''(u)]_n$ et $p(t') = [t''(\varepsilon), \cdots, t''(u)]_n$. On a donc $\mathrm{r}_{\overline{\mathrm{copy}}_n}(p(t''), p(t'))$, et par induction, on a $\mathrm{r}_{o(D')}(p(t), p(t''))$. On a bien $\mathrm{r}_{o(\overline{\mathrm{copy}}_n^1 \cdot D')}(p(t), p(t'))$

Pour le sens inverse, il suffit de faire la traduction dans l'autre sens : étant données une n-pile p et une suite d'opérations o, on leur associe $t(p)$ tel que $p(t(p)) = p$ et $D(o)$ tel que $o(D(o)) = o$, et la démonstration précédente montre qu'on a bien $o(p, p')$ si et seulement si $\mathrm{r}_{D(o)}(t(p), t(p'))$.

On en conclue donc que pour tout GSTRS d'ordre n unaire, il existe un système de réécriture de piles sur $Stacks_n$ équivalent et vice-versa.

□

4.3 Automates d'opérations

Tout comme dans le cas des systèmes de réécriture de piles, il n'existe pas de notion intrinsèque de reconnaissabilité sur les arbres de piles. Cependant, il est possible de définir une telle notion sur les opérations, à l'aide d'automates que nous présentons ici. De même que dans le cas des piles d'ordre supérieur, un ensemble d'arbres de piles est reconnaissables si il est l'image d'un arbre de pile par un ensemble reconnaissable d'opérations. Ces ensembles jouissent de propriétés a priori plus faibles que les ensembles reconnaissables d'opérations sur les piles, notamment nous ne savons pas si ils forment une algèbre de Boole. Cependant, elles conservent la clôture par union et par itération, ce qui nous permettra de démontrer la décidabilité de l'accessibilité sur les graphes associés à des systèmes de réécriture suffixe d'arbres de piles. Nous présentons d'abord la notion d'automates d'opérations, et en dérivons la notion de reconnaissabilité sur les arbres de piles. Ensuite, nous nous intéressons aux propriétés de clôture de ces automates. Nous présentons enfin les systèmes reconnaissables de réécriture d'arbres et une notion de relation

associée aux automates étendant la notion de transducteurs d'arbres, dans le but de calculer la relation d'accessibilité d'un système de réécriture suffixe d'arbre de piles. Enfin nous définissons une forme normalisées de ces automates, similaire à celle définies sur les opérations de piles, qui joueront un rôle crucial dans la preuve de la décidabilité de la $\mathsf{FO}[\xrightarrow{*}]$-théorie d'un graphe associé à un système de réécriture d'arbres de piles présentée au paragraphe suivant.

4.3.1 Définition

Définition 4.7. *Un automate sur $\mathcal{D}_n^*$ est un tuple $A = (Q, \Gamma, I, F, \Delta)$, où*

- *Q est un ensemble fini d'états,*
- *Γ est un alphabet de pile fini,*
- *$I \subseteq Q$ est l'ensemble des états initiaux,*
- *$F \subseteq Q$ est l'ensemble des états finaux,*
- *$\Delta \subseteq \big(Q \times (\mathrm{Ops}_{n-1}(\Gamma) \cup \{\mathrm{copy}_n^1, \overline{\mathrm{copy}}_n^1\}) \times Q\big) \cup ((Q \times Q) \times Q) \cup (Q \times (Q \times Q))$ est l'ensemble des transitions.*

Une opération D est acceptée par A s'il existe un étiquetage de ses sommets par des états de Q tels que les sommets entrants soient étiquetés par des états initiaux, les sommets sortants par des états finaux, et que cet étiquetage soit consistant avec Δ, au sens où pour tous x, y et z respectivement étiquetés par les états p, q et r, et pour tout $\theta \in \mathrm{Ops}_{n-1} \cup \{\mathrm{copy}_n^1, \overline{\mathrm{copy}}_n^1\}$,

$$\begin{aligned} x \xrightarrow{\theta} y &\implies (p, \theta, q) \in \Delta, \\ x \xrightarrow{1} y \wedge x \xrightarrow{2} z &\implies (p, (q, r)) \in \Delta, \\ x \xrightarrow{\bar{1}} z \wedge y \xrightarrow{\bar{2}} z &\implies ((p, q), r) \in \Delta. \end{aligned}$$

On note $\mathrm{Op}(A)$ l'ensemble des opérations reconnues par A. Rec désigne la classe des ensembles d'opérations reconnues par des automates d'opération.

Comme dans le cas des piles d'ordres supérieurs, la notion de reconnaissabilité sur les ensembles d'arbres de piles découle de celle sur les relations.

Définition 4.8. *Un ensemble d'arbres de piles L est dit reconnaissable si il existe un ensemble reconnaissable d'opérations R tel que $L = \{t \mid \exists D \in R, \mathrm{r}_{t_0,D}\}$, où t_0 est l'arbre contenant un unique nœud étiqueté par $[\alpha_0]_{n-1}$ pour $\alpha_0 \in \Gamma$: $t_0 = \{\varepsilon \to [\alpha_0]_{n-1}\}$.*

4.3.2 Propriétés de clôture

Nous montrons ici que Rec est clos par les opérations d'union, d'intersection, d'itération et contient les ensembles finis d'opérations.

Proposition 4.6. *Étant donnés deux automates d'opérations A_1 et A_2, il existe un automate A tel que $\mathrm{Op}(A) = \mathrm{Op}(A_1) \cap \mathrm{Op}(A_2)$*

Démonstration. On construit un automate A qui satisfait la proposition 4.6. Tout d'abord, on s'assure que A_1 et A_2 sont complets en ajoutant un état puit si il existe des transitions manquantes. On construit ensuite l'automate A qui est l'automate produit de A_1 et de A_2 :

$$\begin{aligned} Q &= Q_{A_1} \times Q_{A_2} \\ I &= I_{A_1} \times I_{A_2} \\ F &= F_{A_1} \times F_{A_2} \\ \Delta &= \{((q_1, q_2), \theta, (q_1', q_2')) \mid (q_1, \theta, q_1') \in \Delta_{A_1} \wedge (q_2, \theta, q_2') \in \Delta_{A_2}\} \\ &\cup \{(((q_1, q_2), (q_1', q_2')), (q_1'', q_2'')) \mid ((q_1, q_1'), q_1'') \in \Delta_{A_1} \wedge ((q_2, q_2'), q_2'') \in \Delta_{A_2}\} \\ &\cup \{((q_1, q_2), ((q_1', q_2'), (q_1'', q_2''))) \mid (q_1, (q_1', q_1'')) \in \Delta_{A_1} \wedge (q_2, (q_2', q_2'')) \in \Delta_{A_2}\} \end{aligned}$$

Si une opération admet un étiquetage valide dans A_1 et dans A_2, alors l'étiquetage qui étiquette chaque nœud par les deux états l'étiquetant dans A_1 et dans A_2 est valide. Si une opération admet un étiquetage valide dans A, alors sa restriction aux états de A_1 (resp. A_2), on obtient un étiquetage valide de A_1 (resp. A_2). □

Proposition 4.7. *Étant donnés deux automates d'opérations A_1 et A_2, il existe un automate A tel que* $\mathrm{Op}(A) = \mathrm{Op}(A_1) \cup \mathrm{Op}(A_2)$

Démonstration. On considère l'union disjointe de A_1 et de A_2 :
$Q = Q_{A_1} \uplus Q_{A_2}$
$I = I_{A_1} \uplus I_{A_2}$
$F = F_{A_1} \uplus F_{A_2}$
$\Delta = \Delta_{A_1} \uplus \Delta_{A_2}$
Si une opération admet un étiquetage valide dans A_1 (resp A_2), c'est aussi un étiquetage valide dans A. Si une opération admet un étiquetage valide dans A, comme A est l'union disjointe de A_1 et de A_2, il est forcément uniquement étiqueté par des états de A_1 ou de A_2 (par définition, il n'y a pas de transitions entre des états de A_1 et des états de A_2). Il s'ensuit que c'est un étiquetage valide dans A_1 ou dans A_2. □

Proposition 4.8. *Étant donné un automate d'opération A, il existe A' qui reconnaît* $\mathrm{Op}(A)^*$.

Démonstration. On construit A'.
$Q = Q_A \uplus \{q\}$
$I = I_A \cup \{q\}$
$F = F_A \cup \{q\}$
L'ensemble des transitions Δ contient les transitions de A ainsi que des copies de chaque transitions terminant dans F_A, modifiées pour terminer dans un état de I_A

$$\begin{aligned}
\Delta &= \Delta_A \\
&\cup \{(q_1, \theta, q_i) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, \theta, q_f) \in \Delta_A\} \\
&\cup \{((q_1, q_2), q_i) \mid q_i \in I_A, \exists q_f \in F_A, ((q_1, q_2), q_f) \in \Delta_A\} \\
&\cup \{(q_1, (q_2, q_i)) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, (q_2, q_f)) \in \Delta_A\} \\
&\cup \{(q_1, (q_i, q_2)) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, (q_f, q_2)) \in \Delta_A\} \\
&\cup \{(q_1, (q_i, q_i')) \mid q_i, q_i' \in I_A, \exists q_f, q_f' \in F_A, (q_1, (q_f, q_f')) \in \Delta_A\}
\end{aligned}$$

Pour chaque $k \in \mathbb{N}$, si $D \in (\mathrm{Op}(A)^k)$, il admet un étiquetage valide dans A' : L'opération □ admet un étiquetage valide car q est à la fois initial et final. La propriété est donc vraie pour $(\mathrm{Op}(A)^0)$ Si elle est vraie pour $(\mathrm{Op}(A)^k)$, on prend unne opération G dans $(\mathrm{Op}(A)^{k+1})$ et on la décompose en D de $\mathrm{Op}(A)$ et F de $\mathrm{Op}(A)^k$ (ou symétriquement, $D \in \mathrm{Op}(A)^k$ et $F \in \mathrm{Op}(A)^k$), tel que $G \in D \cdot F$. L'étiquetage qui est l'union de deux étiquetages valides pour D et F et qui étiquette les nœuds identifiés par leur étiquetage dans F (états initiaux) est valide dans A.

Si une opération admet un étiquetage valide dans A', on peut la découper en plusieurs morceaux, en séparant sur les transitions ajoutées, et on obtient une collection d'opérations de $\mathrm{Op}(A)$. On a donc une opération dans un $\mathrm{Op}(A)^k$ pour un certain k. Alors $\mathrm{Op}(A') = \bigcup_{k \geq 0} \mathrm{Op}(A)^k$, et A' reconnaît $\mathrm{Op}(A)^*$. □

Proposition 4.9. *Étant donné une opération D, il existe un automate A tel que* $\mathrm{Op}(A) = \{D\}$.

Démonstration. Si $D = (V, E)$, on prend :
$Q = V$
I est l'ensemble des sommets entrants.
F est l'ensemble des sommets sortants.

$$\begin{aligned}
\Delta &= \{(q, \theta, q') \mid (q, \theta, q') \in E\} \\
&\cup \{(q, (q', q'')) \mid (q, 1, q') \in E \wedge (q, 2, q'') \in E\} \\
&\cup \{((q, q'), q'') \mid (q, 1, q'') \in E \wedge (q', 2, q'') \in E\}
\end{aligned}$$

La partie connexe reconnue est D par construction. □

Ces propriétés de clôture se transfèrent sur les ensembles reconnaissables d'arbres de piles en les appliquant simplement aux automates reconnaissant les opérations les définissant.

Proposition 4.10. *Les ensembles reconnaissables d'arbres de piles sont clos par union et intersection, et contiennent les ensembles finis d'arbres de piles.*

4.3.3 Réécriture reconnaissable et transducteurs suffixes

On étend naturellement la notion de système de réécriture suffixe d'arbres de piles pour définir les systèmes reconnaissables de réécriture suffixe d'arbres de piles en considérant des ensemble R_a reconnaissables.

Définition 4.9. *Un système reconnaissable de réécriture suffixe de n-arbres de piles R étiqueté par Σ est la donnée de $|\Sigma|$ automates d'opérations A_a. Il existe éventuellement un automate A_ε qui définit des transitions silencieuses étiquetées par $\varepsilon \notin \Sigma$. Il induit le graphe $\mathcal{G}_R = (ST_n, E)$ avec $E = \{(t, a, t') \mid \exists D \in \mathrm{Op}(A_a), \mathrm{r}_D(t, t')\}$.*

De même que dans le cas des arbres, la réécriture reconnaissable ne permet pas de rattraper la clôture transitive des systèmes de réécriture suffixe d'arbres de piles, puisqu'elle impose de ne réécrire qu'un sous-arbre en un pas quand dans la clôture transitive, il est possible de réécrire plusieurs sous-arbres simultanément. Dans le cas des arbres, cette difficulté est réglée par l'introduction des transducteurs suffixes d'arbres, dont la seule différence avec un système reconnaissable de réécriture suffixe d'arbre est l'application des règles à un arbre (une seule application dans le cas de la réécriture, un nombre arbitraire d'applications parallèles dans le cas du transducteur). Dans cet esprit, on définit donc la relation reconnue par un automate $\mathcal{A}$, comme $\mathcal{R}(A) = \bigcup_{\vec{D} \in \mathrm{Op}(A)^*} \mathrm{r}_{\vec{D}}$, c'est-à-dire exactement comme l'application parallèle de plusieurs opérations reconnues par A.

Définition 4.10. *Un transducteur suffixe de n-arbres de piles Λ étiqueté par Σ est la donnée de $|\Sigma|$ automates d'opérations A_a. Il existe éventuellement un automate A_ε qui définit des transitions silencieuses étiquetées par $\varepsilon \notin \Sigma$. Il induit le graphe $\mathcal{G}_R = (ST_n, E)$ avec $E = \{(t, a, t') \mid (t, t') \in \mathcal{R}(A_a)\}$.*

À l'ordre 1, nous avons déjà vu que les arbres de piles sont simplement des arbres, et que les systèmes de réécriture suffixe d'arbres de piles coïncident avec les systèmes de réécriture suffixe d'arbres. Il suffit d'étendre la preuve de la propriété 4.2 à des ensemble reconnaissables de règles, on a la propriété suivante.

Proposition 4.11. *La classe des relations reconnues par les automates d'opérations d'ordre* 1 *et celle définie par les transducteurs suffixes d'arbres coïncident.*

Enfin, montrons que les transducteurs d'arbres suffixes rattrapent bien la clôture transitive des systèmes de réécriture suffixe d'arbres de piles.

Proposition 4.12. *Étant donné R un système (fini ou reconnaissable) de réécriture suffixe d'arbres de piles, il existe un automate d'opérations tel que pour tous $t, t' \in ST_n$, $t \xrightarrow[R]{*} t' \Leftrightarrow (t, t') \in \mathcal{R}(A)$.*

Étant donné Λ un transducteur suffixe d'arbres de piles, il existe un automate d'opérations tel que pour tous $t, t' \in ST_n$, $t \xrightarrow[\Lambda]{*} t' \Leftrightarrow (t, t') \in \mathcal{R}(A)$.

Démonstration. Cette proposition est une conséquence des propriétés de clôture du sous-paragraphe précédent. En effet, tout ensemble fini d'opération est reconnu par un automate, et les ensemble reconnaissables sont clos par union, par conséquent, pour tout système (fini ou reconnaissable) de réécriture R, il existe un automate d'opérations A (qui est l'union des automates reconnaissant les opérations étiquetée par chaque lettre) tel que $t \xrightarrow[R]{} t' \Leftrightarrow \exists D \in \mathrm{Op}(A), \mathrm{r}_D(t, t')$,

et pour tout transducteur suffixe d'arbres de piles Λ, il existe un automate d'opérations A (qui est l'union des automates de chaque lettre) tel que $t \xrightarrow[R]{} t' \Leftrightarrow (t, t') \in \mathcal{R}(A)$.

Dans les deux cas, il existe un automate A' qui reconnaît $\mathrm{Op}(A)^*$ (propriété 4.8). Commençons par montrer la proposition pour les systèmes de réécriture. On montre par induction que $t \xrightarrow[R]{*} t' \Leftrightarrow (t, t') \in \mathrm{R}(A')$.

Si $t = t'$, la propriété est évidente, puisque par définition, tous les couples (t, t) sont dans la relation reconnue par A'.

Si il existe t'' tel que $(t'', t') \in \mathrm{R}(A')$ et $t \xrightarrow[R]{} t''$. Par définition, il existe $\vec{D} \in \mathrm{Op}(A)^*$ et $\vec{i} \in \mathbb{N}^*$ tels que $\mathrm{r}^{\vec{i}}_{\vec{D}}(t'', t')$ et $D \in R$ et $j \in \mathbb{N}$ tel que $\mathrm{r}^{j}_{D}(t, t'')$. On a maintenant deux cas possibles :

- Soit D est appliqué de manière disjointe des éléments de $\vec{D}$. Comme D est dans R, il est dans $\mathrm{Op}(A)$ et donc dans $\mathrm{Op}(A')$. Ainsi $\mathrm{r}^{\vec{i'}}_{\vec{D'}}(t, t')$ avec $\vec{D'} = (D_1, \cdots, D_{|\vec{D}|}, D)$ et $\vec{i'} = (i_1, \cdots, i_{|\vec{D}|}, j)$.
- Soit on applique $\vec{D}$ à des feuilles produites par D. Supposons que les opération $D_{j_1}, \cdots, D_{j_\ell}$ sont appliquées à des feuilles produites par D, alors, par définition de l'application, il existe des entiers $k_1, \cdots, k_\ell, h_1, \cdots, h_\ell$ tels que $\mathrm{r}^{i_{j_\ell}}_{D_{j_\ell}} \circ \cdots \circ \mathrm{r}^{i_{j_1}}_{D_{j_1}} \circ \mathrm{r}^{j}_{D} = \mathrm{r}^{\min(j, j_1)}_{D'}$, avec $D' = (\cdots (D \cdot_{k_\ell, h_\ell} D_{j_\ell}) \cdots D_{j_2}) \cdot_{k_1, h_1} D_1$. Informellement, cela revient simplement à concaténer D avec les opérations qui s'applique après lui, à l'endroit correspondant. Le $\min(j, i_{j_1})$ est là pour s'assurer qu'on applique l'opération obtenue au bon endroit, à savoir j si le nœud entrant le plus à gauche de D' est celui de D, ou i_{j_1} si c'est celui de D_{j_1}. D' appartient bien à $\mathrm{Op}(A)^*$, et donc à $\mathrm{Op}(A')$, et ainsi, le vecteur $\vec{D'}$ obtenu en remplaçant les D_{j_k} par D' appliqué aux positions $\vec{i'}$ obtenu en remplaçant les i_k par $\min(j, i_{j_1})$ est bien tel que $\mathrm{r}^{\vec{i'}}_{\vec{D'}} = \mathrm{r}^{\vec{i}}_{\vec{D}} \circ \mathrm{r}^{j}_{D}$.

Ainsi, dans tous les cas, on a bien $(t, t') \in \mathrm{R}(A')$.

L'autre direction se montre de la même manière en montrant que tout vecteur d'opérations reconnues par A' se décompose en des opérations de A appliquées aux endroits définis par le vecteur $\vec{i}$.

Ainsi, $\mathrm{R}(A')$ est bien la clôture transitive de la relation définie par R.

Pour un transducteur suffixe d'arbres de piles, la démonstration est la même en remplaçant l'opération D par un vecteur d'opération. L'écriture en est plus fastidieuse, mais le principe reste inchangé.

□

Grâce à cette propriété, on peut également étendre la propriété 3.4 aux systèmes de réécriture suffixe d'arbres de piles.

Proposition 4.13. *Étant donnés R un système suffixe de réécriture d'arbres de piles ou un transducteur suffixe d'arbres de piles, et L un ensemble reconnaissable d'arbres de piles, l'ensemble $L' = \{t \mid \exists t_0 \in L, t_0 \xrightarrow[R]{*} t\}$ est reconnaissable.*

Pour montrer cette proposition, il suffit de concaténer l'automate reconnaissant les opérations construisant L avec l'automate reconnaissant la clôture transitive de la relation définie par le système de réécriture de la même manière que dans la preuve précédente, et d'observer qu'on construit bien ainsi un automate reconnaissant les opération construisant l'ensemble L'.

4.3.4 Automates normalisés

Nous introduisons maintenant la notion d'opérations normalisées et d'automates normalisés. Cette notion est similaire à celle des opérations de piles normalisées dans le sens où on va chercher à éliminer les «circuits» des opérations en les remplaçant par des tests pour conserver la relation reconnue. D'ailleurs, pour les opérations unaires, la notion d'opération réduite coïncidera

avec celle définie dans le cas des piles. Pour les opérations contenant des opérations d'arbres, on ne pourra pas simplifier de la même manière. En effet, dans le cas des piles, on pouvait simplifier $\overline{\text{copy}}_i\text{copy}_i$ en un test vérifiant que les deux piles de niveau i les plus hautes sont égales. L'opération $\overline{\text{copy}}^2_n \cdot_{1,1} \text{copy}^2_n$ ne peut être simplifiée de la même manière par un test vérifiant qu'un père est égal à ses deux fils qui serait une opération de base unaire, puisque cette opération comporte deux nœuds entrants et deux nœuds sortant. Aussi ces «circuits» seront conservés. Les autres sous-opérations définissant des «circuits» pourront être simplifiées, notamment les opérations de la forme $\text{copy}^2_n \cdot_{1,1} \overline{\text{copy}}^2_n$ est simplifiable en l'opération vide. Cette notion de normalisation jouera un rôle crucial dans la preuve de la décidabilité de la $\mathsf{FO}[\xrightarrow{*}]$-théorie des graphes définis par des systèmes de réécriture suffixe d'arbres de piles au paragraphe suivant.

On définit l'ensemble des suites normalisées d'étiquettes de DAG à partir de la définition des opérations de piles normalisées.

Définition 4.11. $\text{Rew}'_n = (\text{Rew}_{n-1} \cdot \{\bar{1}, \bar{2}\})^* \cdot \text{Rew}_{n-1} \cdot (\{1, 2\} \cdot \text{Rew}_{n-1})^*$.

Définition 4.12. *Une opération avec tests D est* normalisée *si pour tous $x, y \in V_D$, si $x \xrightarrow{w} y$, alors $w \in \text{Rew}'_n$.*

Un automate d'opérations avec tests est dit normalisé s'il reconnait uniquement des opérations normalisées.

La proposition essentielle de la notion d'automate normalisé est que comme dans le cas des piles, pour tout ensemble reconnaissable d'opérations avec tests, il existe un ensemble reconnaissable d'opérations avec tests normalisées définissant la même relation.

Proposition 4.14. *Pour tout automate A, Il existe un automate d'opérations avec tests normalisé distingué A_r tel que $\mathcal{R}(A) = \mathcal{R}(A_r)$ et $\bigcup_{D\in A} \text{r}_D = \bigcup_{D\in A_r} \text{r}_D$.*

Le reste de ce paragraphe est consacré à la preuve de cette proposition. Notons que les deux parties de la propositions diffèrent uniquement par le nombre d'opérations reconnues par l'automate que l'on applique simultanément (un nombre arbitraire ou une seule), aussi, dans la preuve, on ne distinguera pas ces cas, puisqu'ils reviendront simplement à montrer que la relation définie par une opération de l'automate d'origine est incluse dans l'union de relations définies par des opérations normalisées reconnues par l'automate construit, et vice-versa.

Preuve de la propriété 4.14

Remarquons tout d'abord que si on a aucune transition d'arbre, on a un automate d'opérations sur les piles comme défini précédemment.

L'idée de cette preuve est de séparer l'automate en deux parties : l'une contenant les transitions d'arbres et l'autre les transitions de piles, de normaliser chacune de ces parties séparément et d'ensuite retirer les transitions utilisées pour séparer l'automate en deux parties.

Étape 1 Dans cette transformation, nous utilisons une nouvelle opération de base spéciale : id telle que l'opération associée D_{id} est le DAG suivant : $V_{D_{\text{id}}} = \{x, y\}$ et $E_{D_{\text{id}}} = \{(x, \text{id}, y)\}$. Pour tout arbre de pile t et tout entier $i \leq |\text{fr}(t)|$, $\text{id}_{(i)}(t) = t$. Nous utilisons cette opération pour séparer notre opération en plusieurs parties liées par des id, et nous les éliminerons à la fin de la transformation. On suppose que l'automate de départ ne contient pas de transition étiquetée par id.

On commence par découper l'ensemble d'états de contrôles de l'automates en trois parties. On crée trois copies de Q :

- Q_s qui sont les sources et les buts de toutes les transitions de piles, les buts des id-transitions partant de Q_{t_1} et sources des id-transitions arrivant dans Q_{t_2}.

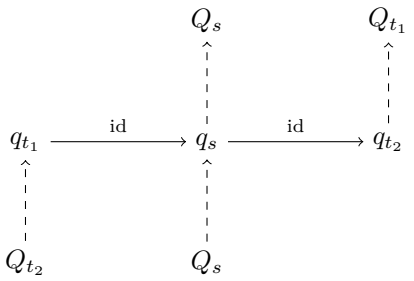


FIGURE 4.8 – Étape 1 : Séparation d'un état q

· Q_{t_1} qui sont les buts de toutes les transitions d'arbre et les sources des id-transitions arrivant dans Q_s.

· Q_{t_2} qui sont les sources de toutes les transitions d'arbres et les buts des id-transitions partant de Q_s.

L'idée de ce que l'on veut obtenir est présenté en figure 4.8.
Formellement, on remplace $A = (Q, I, F, \Delta)$ par $A_1 = (Q', I', F', \Delta')$ où :

$$\begin{array}{lcl}
Q' & = & \{q_{t_1}, q_{t_2}, q_s \mid q \in Q\} \\
I' & = & \{q_s \mid q \in I\} \\
F' & = & \{q_s \mid q \in F\} \\
\Delta & = & \{(q_s, \theta, q'_s) \mid (q, \theta, q') \in \Delta\} \\
 & \cup & \{(q_{t_2}, (q'_{t_1}, q''_{t_1})) \mid (q, (q', q'')) \in \Delta\} \\
 & \cup & \{((q_{t_2}, q'_{t_2}), q''_{t_1}) \mid ((q, q'), q'') \in \Delta\} \\
 & \cup & \{(q_{t_2}, \mathrm{copy}^1_n, q'_{t_1}) \mid (q, \mathrm{copy}^1_n, q') \in \Delta\} \\
 & \cup & \{(q_{t_2}, \overline{\mathrm{copy}}^1_n, q'_{t_1}) \mid (q, \overline{\mathrm{copy}}^1_n, q') \in \Delta\} \\
 & \cup & \{(q_{t_1}, \mathrm{id}, q_s), (q_s, \mathrm{id}, q_{t_2}) \mid q \in Q\}
\end{array}$$

où pour tout $q \in Q$, q_{t_1}, q_{t_2}, q_s sont des états frais.

Lemme 4.15. *A et A_1 reconnaissent la même relation.*

Démonstration. Pour démontrer ce lemme, on prouve que pour chaque opération D reconnue par A, il existe une opération D' reconnue par A_1 telle que $R_D = R_{D'}$, et vice versa.

Soit D une opération reconnue par A. On montre par induction sur la structure de D qu'on peut construire D' telle que $R_D = R_{D'}$ et pour tout étiquetage ρ_D de D compatible avec Δ, avec I_D étiqueté par $\vec{q}$ et O_D par $\vec{q'}$, il existe $\rho_{D'}$ un étiquetage de D' compatible avec Δ' tel que $I_{D'}$ est étiqueté par $\vec{q_s}$ et $O_{D'}$ par $\vec{q'_s}$.

Si $D = \square$, on choisit $D' = \square$. Or a $R_D = R_{D'}$. Pour tout étiquetage ρ_D qui étiquette le seul sommet de D par q, on choisit $\rho_{D'}$ qui étiquette le seul sommet de D' par q_s. Ces étiquetages sont compatibles avec Δ et Δ', par vacuité.

Supposons maintenant qu'on a F et F' telles que pour tout étiquetage ρ_F de F on puisse définir un étiquetage $\rho_{F'}$ de F' satisfaisant la condition précédent. Considérons les cas suivants :

· $D = (F \cdot_{1,1} D_\theta) \cdot_{1,1} G$, pour $\theta \in \{\mathrm{copy}^1_n, \overline{\mathrm{copy}}^1_n\}$. On nomme x le sommet sortant de F et y le sommet d'entrée de G. On a $V_D = V_F \cup V_G$ et $E_D = E_F \cup E_G \cup \{x \xrightarrow{\theta} y\}$.
Par hypothèse d'induction, on considère F' et G', et on construit $D' = (((F' \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} D_\theta) \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} G'$, avec $V_{D'} = V_{F'} \cup V_{G'} \cup \{x'_1, x'_2\}$ et $E_{D'} = E_{F'} \cup E_{G'} \cup \{x' \xrightarrow{\mathrm{id}} x'_1, x'_1 \xrightarrow{\theta} x'_2, x'_2 \xrightarrow{\mathrm{id}} y'\}$, où x' est le sommet sortant de F' et y' le sommet d'entrée de G'.
On choisit ρ_D un étiquetage de D et ρ_F (resp. ρ_G) sa restriction à F (resp. à G). On a $\rho_D(x) = q$ et $\rho_D(y) = q'$. Par hypothèse d'induction, on considère $\rho_{F'}$ (resp. $\rho_{G'}$)

L'étiquetage correspondant de F' (resp. G'), avec $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y') = q'_s$). Nous construisons alors $\rho_{D'} = \rho_{F'} \cup \rho_{G'} \cup \{x'_1 \to q_{t_2}, x'_2 \to q'_{t_1}\}$.
Comme ρ_D est compatible avec Δ, (q, θ, q') appartient à Δ, et donc, par on construition, $(q_{t_2}, \theta, q'_{t_1})$ appartient à Δ'. On a également que $(q_s, \mathrm{id}, q_{t_2})$ et $(q'_{t_1}, \mathrm{id}, q'_s)$ appartiennent à Δ'. Alors, ρ'_D est compatible avec Δ'.
Pour montrer que $\mathrm{r}_D = \mathrm{r}_{D'}$, il suffit de remarquer que, par définition de l'application d'une opération, pour tout entier i, on a $\mathrm{r}^i_{D'} = \mathrm{r}^i_{F'} \circ \mathrm{id}^i \circ \mathrm{r}^i_\theta \circ \mathrm{id}^i \circ \mathrm{r}^i_{G'} = \mathrm{r}^i_F \circ \mathrm{r}^i_\theta \circ \mathrm{r}^i_G = \mathrm{r}^i_D$.

Les autres cas étant similaires, dans la suite nous donnons seulement D' et $\rho_{D'}$, et laissons les détails au lecteur.

- $D = (F \cdot_{1,1} D_\theta) \cdot_{1,1} G$, pour $\theta \in \mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1}$. On appelle x le sommet sortant de F et y le sommet d'entrée de G. On a $V_D = V_F \cup V_G$ et $E_D = E_F \cup E_G \cup \{x \xrightarrow{\theta} y\}$.
Par hypothèse d'induction, on considère F' et G', et on construit $D' = (F' \cdot_{1,1} \theta) \cdot_{1,1} G'$, avec $V_{D'} = V_{F'} \cup V_{G'}$ et $E_{D'} = E_{F'} \cup E_{G'} \cup \{x' \xrightarrow{\theta} y'\}$, où x' est le sommet sortant de F' et y' le sommet d'entrée de G'.
On choisit ρ_D un étiquetage de D et ρ_F (resp. ρ_G) sa restriction à F (resp. à G). On a $\rho_D(x) = q$ et $\rho_D(y) = q'$. Par hypothèse d'induction, on considère $\rho_{F'}$ (resp. $\rho_{G'}$) l'étiquetage correspondant de F' (resp. G'), avec $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y') = q'_s$). Ensuite, on construit $\rho_{D'} = \rho_{F'} \cup \rho_{G'}$.
- $D = ((F \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} H) \cdot_{1,1} G$. On appelle x le sommet sortant de F, y le sommet d'entrée de G et z le sommet d'entrée de H. On a $V_D = V_F \cup V_G \cup V_H$ et $E_D = E_F \cup E_G \cup E_H \cup \{x \xrightarrow{1} y, x \xrightarrow{2} z\}$.
Par hypothèse d'induction, on considère F', G' et H', et on construit $D' = (((((F \cdot_{1,1} D_{\mathrm{id}}) D_{\mathrm{copy}^2_n}) \cdot_{2,1} D_{\mathrm{id}}) \cdot_{2,1} H) \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} G$, avec $V_{D'} = V_{F'} \cup V_{G'} \cup V_{H'} \cup \{x'_1, x'_2, x'_3\}$ et $E_{D'} = E_{F'} \cup E_{G'} \cup E_{H'} \{x' \xrightarrow{\mathrm{id}} x'_1, x'_1 \xrightarrow{1} x'_2, x'_1 \xrightarrow{2} x'_3, x'_2 \xrightarrow{\mathrm{id}} y', x'_3 \xrightarrow{\mathrm{id}} z'\}$, où x' est le sommet sortant de F', y' le sommet d'entrée de G' et z' le sommet d'entrée de H'.
On choisit ρ_D un étiquetage de D et ρ_F (resp. ρ_G, ρ_H) sa restriction à F (resp. à G, à H). On a $\rho_D(x) = q$, $\rho_D(y) = q'$ et $\rho_D(z) = q''$. Par hypothèse d'induction, on considère $\rho_{F'}$ (resp. $\rho_{G'}$,$\rho_{H'}$) l'étiquetage correspondant de F' (resp. G',H'), avec $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y') = q'_s$, $\rho_{H'}(z') = q''_s$). On construit alors $\rho_{D'} = \rho_{F'} \cup \rho_{G'} \cup \rho_{H'} \cup \{x'_1 \to q_{t_2}, x'_2 \to q'_{t_1}, x'_3 \to q''_{t_1}\}$.
- $D = (F \cdot_{1,1} (G \cdot_{1,2} D_{\overline{\mathrm{copy}^2_n}})) \cdot_{1,1} H$. On appelle x le sommet sortant de F, y le sommet sortant de G et z le sommet d'entrée de H. On a $V_D = V_F \cup V_G \cup V_H$ et $E_D = E_F \cup E_G \cup E_H \cup \{x \xrightarrow{\bar{1}} z, y \xrightarrow{\bar{2}} z\}$.
Par hypothèse d'induction, on considère F', G' et H', et on construit $D' = (((F \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} ((G \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,2} D_{\overline{\mathrm{copy}^2_n}})) \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} H$, avec $V_{D'} = V_{F'} \cup V_{G'} \cup V_{H'} \cup \{x'_1, x'_2, x'_3\}$ et $E_{D'} = E_{F'} \cup E_{G'} \cup E_{H'} \{x' \xrightarrow{\mathrm{id}} x'_1, y' \xrightarrow{\mathrm{id}} x'_2, x'_1 \xrightarrow{\bar{1}} x'_3, x'_2 \xrightarrow{\bar{2}} x'_3, x'_3 \xrightarrow{\mathrm{id}} z'\}$, où x' est le sommet sortant de F', y' le sommet d'entrée de G' et z' le sommet d'entrée de H'.
On choisit ρ_D un étiquetage de D et ρ_F (resp. ρ_G, ρ_H) sa restriction à F (resp. à G, à H). On a $\rho_D(x) = q$, $\rho_D(y) = q'$ et $\rho_D(z) = q''$. Par hypothèse d'induction, on considère $\rho_{F'}$ (resp. $\rho_{G'}$, $\rho_{H'}$) l'étiquetage correspondant de F' (resp. G', H'), avec $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y') = q'_s$, $\rho_{H'}(z') = q''_s$). On construit alors $\rho_{D'} = \rho_{F'} \cup \rho_{G'} \cup \rho_{H'} \cup \{x'_1 \to q_{t_2}, x'_2 \to q'_{t_2}, x'_3 \to q''_{t_1}\}$.
- $D = (((((F \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} H) \cdot_{1,1} G) \cdot_{1,1} D_{\overline{\mathrm{copy}^2_n}}) \cdot_{1,1} K$. On appelle x le sommet sortant de F, y_1 le sommet d'entrée de G et y_2 son sommet sortant, z_1 le sommet d'entrée de H et z_2 son sommet sortant et w le sommet d'entrée de K. On a $V_D = V_F \cup V_G \cup V_H \cup V_K$ et $E_D = E_F \cup E_G \cup E_H \cup E_K \cup \{x \xrightarrow{1} y_1, x \xrightarrow{2} z_1, y_2 \xrightarrow{\bar{1}} t, z_2 \xrightarrow{\bar{2}} t\}$.
Par hypothèse d'induction, on considère F', G', H' et K', et on construit $D' = ((((((F' \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} (D_{\mathrm{id}} \cdot_{1,1} H')) \cdot_{1,1} (D_{\mathrm{id}} \cdot_{1,1} G')) \cdot_{1,1} D_{\overline{\mathrm{copy}^2_n}}) \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} K'$, avec $V_{D'} =$

$V_{F'} \cup V_{G'} \cup V_{H'} \cup V_{K'} \cup \{x'_1, x'_2, x'_3, x'_4, x'_5, x'_6\}$ et $E_{D'} = E_{F'} \cup E_{G'} \cup E_{H'} \cup E_{K'}\{x' \xrightarrow{\text{id}} x'_1, x'_1 \xrightarrow{1} x'_2, x'_1 \xrightarrow{2} x'_3, x'_2 \xrightarrow{\text{id}} y'_1, x'_3 \xrightarrow{\text{id}} z'_1, y'_2 \xrightarrow{\text{id}} x'_4, z'_2 \xrightarrow{\text{id}} x'_5, x'_4 \xrightarrow{\bar{1}} x'_6, x'_5 \xrightarrow{\bar{2}} x'_6, x'_6 \xrightarrow{\text{id}} t'\}$, où x' is le sommet sortant de F', y'_1 le sommet d'entrée de G', y'_2 son sommet sortant, z'_1 le sommet d'entrée de H', z'_2 son sommet sortant et t' le sommet d'entrée de K'.

On choisit ρ_D un étiquetage de D_D et ρ_F (resp. ρ_G, ρ_H, ρ_K) sa restriction à F (resp. G, H, K). On a $\rho_D(x) = q$, $\rho_D(y_1) = q'$, $\rho_D(z_1) = q''$, $\rho_D(y_2) = r'$, $\rho_D(z_2) = r''$ et $\rho_D(t) = r''$. Par hypothèse d'induction, on considère $\rho_{F'}$ (resp. $\rho_{G'}$, $\rho_{H'}$, $\rho_{K'}$) l'étiquetage correspondant de F' (resp. G', H', K'), avec $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y'_1) = q'_s$, $\rho_{H'}(z'_1) = q''_s$, $\rho_{G'}(y'_2) = r'_s$, $\rho_{H'}(z'_2) = r''_s$, $\rho_{K'}(t') = r''_s$). On construit alors $\rho_{D'} = \rho_{F'} \cup \rho_{G'} \cup \rho_{H'} \cup \{x'_1 \to q_{t_2}, x'_2 \to q'_{t_1}, x'_3 \to q''_{t_1}, x'_4 \to r_{t_2}, x'_5 \to r'_{t_2}, x'_6 \to r''_{t_1}\}$.

Pour l'autre sens, on choisit D' reconnu par A_1 et on montre qu'on peut construire D reconnue par A avec $R_D = R_{D'}$ par une induction sur la structure de D' similaire à la précédente (pour chaque id-transition, on ne modifie pas l'opération construite et pour toute autre transition, on l'ajoute à l'opération construite). Tous les arguments sont similaires à ceux de la preuve précédente et nous en laissons donc les détails au lecteur.

□

Dans la suite, on commence par normaliser la partie contenant les transitions d'arbre de l'automate. Pour ce faire, il suffit d'empêcher l'automate de reconnaître des opérations contenant $((D_{\text{copy}^2_n} \cdot_{1,1} F_1) \cdot_{2,1} F_2) \cdot_{1,1} D_{\overline{\text{copy}}^2_n}$, ou $(D_{\text{copy}^1_n} \cdot_{1,1} F) \cdot_{1,1} D_{\overline{\text{copy}}^1_n}$ comme sous-opération. Une telle sous-opération est appelée une bulle. Toutefois, on ne veut pas que l'opération reconnue soit modifiée. Nous le faisons en deux étapes : d'abord en autorisant l'automate à remplacer ces bulles par des tests équivalents (après avoir remarqué qu'une bulle ne peut définir qu'un test) dans toute opération reconnue (étape 2), et ensuite en s'assurant qu'il ne peut y avoir de $\overline{\text{copy}}^i_n$-transition qui suit la première copy^j_n transition (étape 3).

Étape 2 Soit $A_1 = (Q, I, F, \Delta)$ l'automate obtenu après l'étape 1. Étant donnés deux états q_1, q_2, on note $L_{A_{q_1,q_2}}$ l'ensemble $\{s \in \mathit{Stacks}_{n-1} \mid \exists D \in \mathcal{D}(A_1), D_{(1)}(s) = s\}$ où A_{q_1,q_2} est une copie de A_1 dans laquelle on prend q_1 comme unique état initial et q_2 comme unique état final. En d'autres mots, $L_{A_{q_1,q_2}}$ est l'ensemble des $(n-1)$-piles tel que les arbres avec un nœud étiqueté par cette pile restent inchangés par une opération reconnue par A_{q_1,q_2}. On définit $A_2 = (Q, I, F, \Delta')$ avec

$$\begin{array}{rcl} \Delta' & = & \Delta \\ & \cup & \{(q_s, T_{L_{A_{r_s,r'_s}} \cap L_{A_{s_s,s'_s}}}, q'_s) \mid (q_{t_2}, (r_{t_1}, s_{t_1})), ((r'_{t_2}, s'_{t_2}), q'_{t_1}) \in \Delta\} \\ & \cup & \{(q_s, T_{L_{r_s,s'_s}}, q'_s \mid (q_{t_2}, \text{copy}^1_n, r_{t_1}), (r'_{t_2}, \overline{\text{copy}}^1_n, q'_{t_1}) \in \Delta\} \end{array}$$

La figure 4.9 illustre l'idée de cette construction.

On donne le lemme suivant pour les bulles binaires. Le cas des bulles unaires est très similaire et est de ce fait laissé au lecteur.

Lemme 4.16. *Soient $C_1 = (Q_{C_1}, \{i_{C_1}\}, \{f_{C_1}\}, \Delta_{C_1})$ et $C_2 = (Q_{C_2}, \{i_{C_2}\}, \{f_{C_2}\}, \Delta_{C_2})$ deux automates reconnaissant des opérations sans opérations d'arbres. Les deux automates $B_1 = (Q_1, I, F, \Delta_1)$ et $B_2 = (Q_2, I, F, \Delta_2)$, avec $I = \{q_1\}$, $F = \{q_2\}$, $Q_1 = \{q_1, q_2\}$, $\Delta_1 = \{(q_1, T_{L_{C_1} \cap L_{C_2}}, q_2)\}$, $Q_2 = \{q_1, q_2\} \cup Q_{C_1} \cup Q_{C_2}$ et $\Delta_2 = \{(q_1, (i_{C_1}, i_{C_2})), ((f_{C_1}, f_{C_2}), q_2)\} \cup \Delta_{C_1} \cup \Delta_{C_2}$ reconnaissent les mêmes relations.*

Démonstration. Une opération D reconnue par B_2 est de la forme $D = D_{\text{copy}^2_n} \cdot_{1,1} (F_1 \cdot_{1,1} (F_2 \cdot_{2,2}$

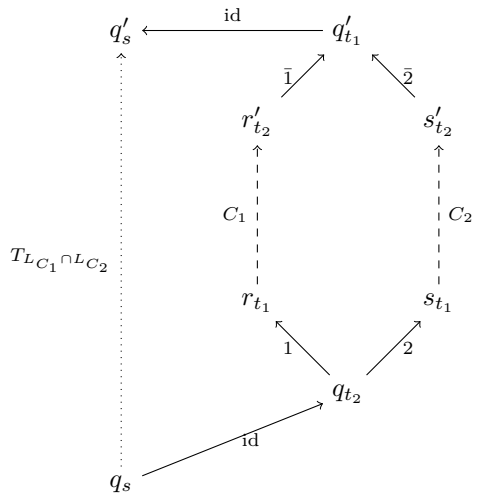


FIGURE 4.9 – Étape 2 : La transition étiquetée par un test ajoutée pour court-circuiter la bulle est représentée par une ligne pointillée

$D_{\overline{\mathrm{copy}}^2_n}$)), où F_1 est reconnue par C_1 et F_2 par C_2. On a :

$$\begin{aligned}
\mathrm{r}^i_D(t_1,t_2) \Leftrightarrow & \mathrm{r}^i_{\overline{\mathrm{copy}}^2_n} \circ \mathrm{r}^{i+1}_{F_2} \circ \mathrm{r}^i_{F_1} \circ \mathrm{r}^i_{\mathrm{copy}^2_n}(t_1,t_2) \\
\Leftrightarrow & \exists t_3,t_4,t_5, \mathrm{r}^i_{\mathrm{copy}^2_n}(t_1,t_3) \wedge \mathrm{r}^i_{F_1}(t_3,t_4) \wedge \mathrm{r}^{i+1}_{F_2}(t_4,t_5) \wedge \mathrm{r}^i_{\overline{\mathrm{copy}}^2_n}(t_5,t_2) \\
\Leftrightarrow & \exists t_3,t_4,t_5, \mathrm{dom}(t_3) = \mathrm{dom}(t_1) \cup \{u_i1, u_i2\} \wedge \\
& t_1(u_i) = t_3(u_i) = t_3(u_i1) = t_3(u_i2) \wedge \mathrm{r}^i_{F_1}(t_3,t_4) \wedge \mathrm{r}^{i+1}_{F_2}(t_4,t_5) \wedge \\
& \mathrm{dom}(t_5) = \mathrm{dom}(t_2) \cup \{u_i1, u_i2\} \wedge t_2(u_i) = t_5(u_i) = t_5(u_i1) = t_5(u_i2).
\end{aligned}$$

Par construction, F_1 et F_2 s'appliquent à une unique feuille et ne peuvent pas modifier le père du nœud où ils s'appliquent. On a donc $t_3(u_i) = t_4(u_i) = t_5(u_i)$. La relation est donc non-vide si et seulement si on a $t_3(u_i1) = t_4(u_i1) = t_5(u_i1) = t_3(u_i2) = t_4(u_i2) = t_5(u_i2)$. Ainsi, B_2 accepte uniquement des opérations qui sont des tests, et ces tests sont l'intersection des tests reconnus par C_1 et C_2. La relation reconnue par B_2 est donc exactement la relation reconnue par $T_{L_{C_1} \cap L_{C_2}}$, qui est l'unique opération reconnue par B_1. □

On a le corollaire suivant comme conséquence directe de ce lemme.

Corollaire 4.17. *A_1 et A_2 reconnaissent la même relation.*

En effet, les nouvelles opérations reconnues ne modifient pas la relation reconnue par l'automate, étant donné que chaque test ajouté était déjà présent dans les opérations contenant une bulle.

Étape 3 On prend $A_2 = (Q, I, F, \Delta)$ l'automate obtenu après l'étape 2. Nous voulons maintenant réellement interdire les bulles. Pour ce faire, on divise les états de contrôle de l'automate en deux parties : on crée deux copies de Q :

- Q_d qui ne sont source d'aucune transition étiquetée par copy^d_n,
- Q_c qui ne sont source d'aucune transition étiquetée par $\overline{\mathrm{copy}}^d_n$.

On construit $A_3 = (Q', I', F', \Delta')$ avec :

$$
\begin{array}{lcl}
Q' & = & \{q_d, q_c \mid q \in Q\} \\
I' & = & \{q_d, q_c \mid q \in I\} \\
F' & = & \{q_d, q_c \mid q \in F\} \\
\Delta' & = & \{(q_d, \theta, q'_d), (q_c, \theta, q'_c) \mid (q, \theta, q') \in \Delta, \theta \in \mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1} \cup \{\mathrm{id}\}\} \\
 & \cup & \{((q_d, q'_d), q''_d) \mid ((q, q'), q'') \in \Delta\} \\
 & \cup & \{(q_d, \overline{\mathrm{copy}}^1_n, q'_d) \mid (q, \overline{\mathrm{copy}}^1_n, q') \in \Delta\} \\
 & \cup & \{(q_c, (q'_c, q''_c)), (q_d, (q'_c, q''_c)) \mid (q, (q', q'')) \in \Delta\} \\
 & \cup & \{(q_c, \mathrm{copy}^1_n, q'_c), (q_d, \mathrm{copy}^1_n, q'_c) \mid (q, \mathrm{copy}^1_n, q') \in \Delta\}
\end{array}
$$

Lemme 4.18. *A_2 et A_3 reconnaissent la même relation*

Démonstration. A_3 reconnaît les opérations reconnues par A_2 qui ne contiennent pas de bulles. En effet, tout étiquetage d'une telle opération dans A_2 peut être modifiée pour être un étiquetage dans A_3 (laissé au lecteur). Inversement, toute opération reconnue par A_3 est aussi reconnue par A_2.

On choisit D reconnue par A_2 qui contient au moins une bulle. Supposons que D contienne une bulle F et que $D = D[F]_x$ où D est une opération contenant une bulle de moins et on obtient D en remplaçant le sommet x par F in D. D'après l'étape 2, il existe quatre états de A_2, r_s, r'_s, s_s, s'_s tel que $G = D[T_{L_{A_{r_s,r'_s}} \cap L_{A_{s_s,s'_s}}}]_x$ est reconnu par A_2. Alors $R_D \subseteq R_G$, et G a une bulle de moins que D.

En itérant ce procédé, on obtient une opération D' sans aucune bulle telle que $R_D \subseteq R_{D'}$ et D' est reconnu par A_2. Comme elle ne contient aucune bulle, elle est aussi reconnue par A_3.

Ainsi, toute relation reconnue par une opération avec des bulles est déjà incluse dans la relation reconnue par une opération sans bulle. Ainsi, A_2 et A_3 reconnaissent la même relation. □

On appelle *partie destructive* la restriction $A_{3,d}$ de A_3 à Q_d et *partie constructive* sa restriction $A_{3,c}$ à Q_c.

Étape 4 On considère l'automate A_3 obtenu après l'étape précédente. Observons que dans les deux étapes précédente, nous n'avons pas modifié la séparation entre Q_{t_1}, Q_{t_2} et Q_s. On appelle $A_{3,s}$ la restriction de A_3 à Q_s.

On veut maintenant normaliser $A_{3,s}$. Comme cette partie de l'automate contient uniquement des transitions étiquetées par des opérations de $\mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1}$, on peut le considérer comme un automate d'opérations de piles d'ordre supérieur. On utilise donc le processus de normalisation sur les opérations de piles d'ordre supérieur de [Car05] rappelé au théorème 3.8. Pour tout couple (q_s, q'_s) d'états de Q_s, on construit l'automate normalisé A_{q_s,q'_s} de A' où A' est une copie de $A_{3,s}$ où $I_{A'} = \{q_s\}$ et $F_{A'} = \{q'_s\}$. On suppose que ces automates sont distingués, c'est-à-dire que les états de $I_{A_{q_s,q'_s}}$ ne sont la cible d'aucune transition et que les états de $F_{A_{q_s,q'_s}}$ ne sont la source d'aucune transition. On suppose de plus qu'il n'y a jamais deux transitions étiquetées par des tests l'une à la suite de l'autre (ce n'est pas une supposition forte, une telle suite n'étant pas normalisée au sens de [Car05], mais il est utile de le noter).

On remplace $A_{3,s}$ par l'union de tous les A_{q_s,q'_s} : on définit $A_4 = (Q', I', F', \Delta')$ comme suit :

$$
\begin{array}{lcl}
Q' & = & Q_{t_1} \cup Q_{t_2} \cup \bigcup_{q_s,q'_s} Q_{A_{q_s,q'_s}} \\
I' & = & \bigcup_{q_s \in I, q'_s \in Q_s} I_{A_{q_s,q'_s}} \\
F' & = & \bigcup_{q_s \in Q_s, q'_s \in F} F_{A_{q_s,q'_s}} \\
\Delta' & = & \{K \in \Delta \mid K = (q, (q', q'')) \vee K = ((q, q'), q'') \vee K = (q, \mathrm{copy}^1_n, q') \\
 & & \vee K = (q, \overline{\mathrm{copy}}^1_n, q')\} \\
 & \cup & \bigcup_{q_s,q'_s \in Q_s} \Delta_{A_{q_s,q'_s}} \\
 & \cup & \{(q_{t_1}, \mathrm{id}, i) \mid (q_{t_1}, \mathrm{id}, q'_s) \in \Delta, i \in \bigcup_{q''_s \in Q} I_{A_{q'_s,q''_s}}\} \\
 & \cup & \{(f, \mathrm{id}, q_{t_2}) \mid (q'_s, \mathrm{id}, q_{t_2}) \in \Delta, f \in \bigcup_{q''_s \in Q} F_{A_{q''_s,q'_s}}\} \\
 & \cup & \{(q_{t_1}, \mathrm{id}, f) \mid (q_{t_1}, \mathrm{id}, q'_s) \in \Delta, f \in \bigcup_{q''_s \in Q} F_{A_{q''_s,q'_s}}\} \\
 & \cup & \{(i, \mathrm{id}, q_{t_2}) \mid (q'_s, \mathrm{id}, q_{t_2}) \in \Delta, i \in \bigcup_{q''_s \in Q} I_{A_{q'_s,q''_s}}\}
\end{array}
$$

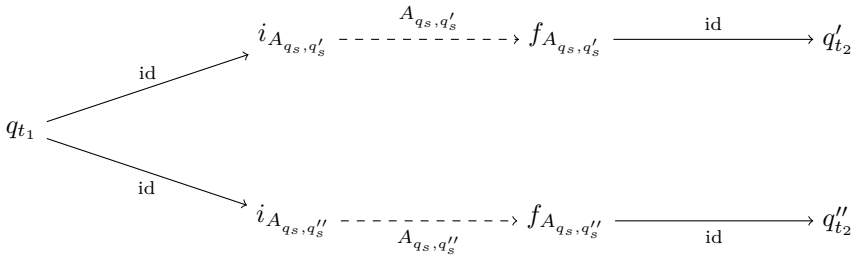


FIGURE 4.10 – Étape 4 : La normalisation de la partie à piles de l'automate

Lemme 4.19. *A_3 et A_4 reconnaissent la même relation.*

Démonstration. Comme A_4 est obtenu en normalisant les parties contenant uniquement des opérations de $\mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1}$ de A_3, la relation définie par les opérations de $(\mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1})^*$ reconnues entre deux états q_s et q'_s est la même pour A_3 et A_4 (théorème 3.8). Pour tout couple (t, t') appartenant à la relation définie par A_3, il existe une opération θ reconnue par A_3 telle que $\mathrm{r}_\theta(t, t')$. Comme le reste de l'automate est inchangé, il est possible de modifier θ en remplaçant chaque partie reconnue entre un q_s et un q'_s par une suite normalisée d'opérations de $(\mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1})^*$ reconnue par A_{q_s,q'_s} et obtenir ainsi une opération θ' reconnue par A_4 telle que $\mathrm{r}_{\theta'}(t, t')$. Ainsi (t, t') est également dans la relation reconnue par A_4. L'autre direction se montre de la même manière. □

Étape 5 Nous avons maintenant un automate normalisé au sens de notre définition, excepté que nous avons des id-transitions. On les élimine via un mécanisme de saturation classique. Observons que dans toutes les étapes précédentes, nous n'avons jamais modifié la séparation entre Q_{t_1}, Q_s et Q_{t_2}, donc toutes les id-transitions vont de Q_{t_1} à Q_s et de Q_s à Q_{t_2}. On prend $A_4 = (Q, I, F, \Delta)$ obtenu après l'étape précédente. On construit $A_5 = (Q', I', F', \Delta')$ avec $Q' = Q_s$, $I' = I$, $F' = F$ et

$$
\begin{aligned}
\Delta' = \Delta \setminus \{&(q, \mathrm{id}, q') \in \Delta\} \\
&\cup \{(q_s, \mathrm{copy}^1_n, q'_s) \mid \exists q''_{t_2}, q'''_{t_1}, (q''_{t_2}, \mathrm{copy}^1_n, q'''_{t_1}), (q'''_{t_1}, \mathrm{id}, q'_s), (q_s, \mathrm{id}, q''_{t_2}) \in \Delta\} \\
&\cup \{(q_s, \overline{\mathrm{copy}}^1_n, q'_s) \mid \exists q''_{t_2}, q'''_{t_1}, (q''_{t_2}, \overline{\mathrm{copy}}^1_n, q'''_{t_1}), (q'''_{t_1}, \mathrm{id}, q'_s), (q_s, \mathrm{id}, q''_{t_2}) \in \Delta\} \\
&\cup \{(q_s, (q'_s, q''_s)) \mid \exists q_1, q_2, q_3, (q_1, (q_2, q_3)), (q_s, \mathrm{id}, q_1), (q_2, \mathrm{id}, q'_s), (q_3, \mathrm{id}, q''_s) \in \Delta\} \\
&\cup \{((q_s, q'_s), q''_s) \mid \exists q_1, q_2, q_3, ((q_1, q_2), q_3), (q_s, \mathrm{id}, q_1), (q'_s, \mathrm{id}, q_2), (q_3, \mathrm{id}, q''_s) \in \Delta\}
\end{aligned}
$$

Lemme 4.20. *A_4 et A_5 reconnaissent la même relation.*

Démonstration. On montre ce lemme par une induction sur la structure des opérations reconnue, similaire à l'induction de l'étape 1. Nous la laissons donc au lecteur. □

Étape 6 On sépare maintenant les états en deux parties :

- Q_T, les états qui sont les cibles de toutes les transitions étiquetées par des tests et source d'aucune transition étiquetée par un test,
- Q_C, les états qui sont les sources de toutes les transitions étiquetées par des tests et cible d'aucune transition étiquetée par un test.

Étant donné l'automate $A_5 = (Q, I, F, \Delta)$ obtenu ci-avant, on définit $A_6 = (Q', I', F', \Delta')$.

$$
\begin{aligned}
Q' &= \{q_T, q_C \mid q \in Q\},\\
I' &= \{q_C \mid q \in I\},\\
F' &= \{q_T, q_C \mid q \in F\},\\
\Delta' &= \{(q_C, \theta, q'_C), (q_T, \theta, q'_C) \mid (q, \theta, q') \in \Delta, \theta \in \mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1}\{\mathrm{copy}^1_n, \overline{\mathrm{copy}}^1_n\}\}\\
&\quad \cup \{((q_C, q'_C), q''_C), ((q_C, q'_T), q''_C), ((q_T, q'_C), q''_C), ((q_T, q'_T), q''_C) \mid ((q, q'), q'') \in \Delta\}\\
&\quad \cup \{(q_C, (q'_C, q''_C)), (q_T, (q'_C, q''_C)) \mid (q, (q', q'')) \in \Delta\}\\
&\quad \cup \{(q_C, T_L, q'_T) \mid (q, T_L, q') \in \Delta\}.
\end{aligned}
$$

Lemme 4.21. *A_5 et A_6 reconnaissent la même relation.*

Démonstration. Étant donné, d'après l'étape 4, qu'il est impossible d'avoir deux transitions étiquetées par des tests de suite, l'ensemble d'opérations reconnues est le même dans les deux automates, seul leur étiquetage étant modifié. Les détails sont laissés au lecteur. □

Finalement, on suppose que l'automate obtenu par ce procédé est distingué, c'est-à-dire que les états initiaux ne sont la cible d'aucune transition et que les états finaux ne sont la source d'aucune transition. Si ce n'est pas le cas, on peut distinguer l'automate par le procédé classique utilisé pour distinguer les automates de mots sans changer le langage reconnu. Nous avons maintenant un automate normalisé avec tests A_6 obtenu après l'application des six étapes reconnaissant la même relation que l'automate initial A. Dans la suite, on considérera les sous-ensembles d'états Q_T, Q_C, Q_d, Q_c comme définis dans les étapes 3 et 6, et $Q_{u,d} = Q_u \cap Q_d$ avec $u \in \{T, C\}$ et $d \in \{d, c\}$.

4.4 Graphes de réécriture d'arbres de piles

Nous avons maintenant tous les éléments en main pour montrer le résultat technique principal de ce chapitre, à savoir que les graphes liés au systèmes de réécriture suffixes d'arbres de piles ont une $\mathsf{FO}[\rightarrow^*]$-théorie décidable. Nous souhaitons étendre les propriétés présenté au paragraphe 3.3, c'est-à-dire définir des graphes similaires aux graphes de réécriture de piles et aux graphes de transducteurs suffixes d'arbres. On définira donc les graphes associés aux systèmes de réécriture suffixe d'arbres de piles et aux automates d'opérations. Commençons par définir formellement les classes de graphes sur lesquelles nous allons montrer ce résultat. Tous comme les graphes du paragraphe 3.3, les graphes que nous considérerons seront restreints à des ensembles accessibles de sommets, car considérer des restrictions différentes nous ferait perdre la décidabilité de l'accessibilité.

Étant donné un système de réécriture de piles R, on définit son graphe associé $\mathcal{G}_R = (V, E)$ avec :

- $V = ST_n(\Gamma)$,
- $E = \{(t, a, t') \mid \exists (\theta, a) \in R, \mathrm{r}_\theta(t, t')\}$.

On considère la restriction de $\mathcal{G}_R$ à l'ensemble de sommets accessibles par R depuis un ensemble reconnaissable de sommets L en remplaçant V par $\{t \mid \exists t_0 \in L, t_0 \xrightarrow[R]{*} t\}$. Le graphe ainsi obtenu est appelé $\mathcal{G}_{R,L}$. On appelle GSTR_n l'ensemble des restrictions à des ensembles accessibles de sommets des graphes de systèmes finis de réécriture suffixe de n-arbres de piles, et RGSTR_n ceux des systèmes reconnaissables.

Étant donné Λ un transducteur suffixe d'arbres de piles, on définit le graphe $\mathcal{G}_\Lambda = (V, E)$ associé à ce transducteur avec :

- $V = ST_n(\Gamma)$.
- $E = \{(t, a, t') \mid (t, t') \in \mathcal{R}(\mathcal{A}_a)\}$.

De même que précédemment, on considère la restriction à l'ensemble des sommets accessibles par Λ depuis un ensemble reconnaissable de sommets. Le graphe ainsi obtenu est appelé $\mathcal{G}_{\Lambda,L}$ On appelle GSTT_n l'ensemble des restrictions à des ensembles accessibles de sommets des graphes associés à des automates d'arbres.

Comme nous l'avons vu au paragraphe précédent la clôture transitive d'un système de réécriture suffixe d'arbres de piles est définie par un automate d'opérations, de même que la clôture transitive d'une relation définie par un automate d'opérations. Ainsi la clôture transitive d'un graphe de GSTR_n est un graphe de GSTT_n, et cela vaut également pour les graphes de RGSTR_n et de GSTT_n.

Théorème 4.22. *Les graphes de* GSTR_n, RGSTR_n *et* GSTT_n *possèdent une* $\mathsf{FO}[\xrightarrow{*}]$*-théorie décidable.*

Pour montrer ce théorème, on va utiliser une remarque de Colcombet et Löding dans [CL07], qui dit que si un graphe est obtenu par une interprétation à ensembles finis (FSI) à partir d'un graphe ayant une MSO-théorie décidable possède une FO-théorie décidable, en l'appliquant à un graphe des configurations d'un système reconnaissable de réécriture d'arbres de piles auquel on a ajouté la relation d'accessibilité du système, qui est définie par un automate d'opérations. Le théorème 4.22 découle donc de la propriété suivante.

Proposition 4.23. *Étant donnés un système reconnaissable de réécriture suffixe de n-arbres de piles* Σ*-étiqueté* R *et un ensemble reconnaissable de n-arbres de piles* L*, il existe une interprétation à ensemble finis* $(\delta, \phi_{a_1}, \cdots, \phi_{a_k}, \phi_\tau)$ *depuis* $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$ *la structure arborescente d'ordre* n *de* $\Gamma \times \{\varepsilon, 11, 21, 22\}$ *dans le graphe* $\mathcal{G}_{R,L}$*, auquel on a ajouté la relation d'accessibilité de* R*.*

Étant donnés Λ *un transducteur suffixe d'arbres de piles et un ensemble reconnaissable de n-arbres de piles* L*, il existe une interprétation à ensemble finis* $(\delta, \phi_{a_1}, \cdots, \phi_{a_k}, \phi_\tau)$ *depuis* $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$ *la structure arborescente d'ordre* n *de* $\Gamma \times \{\varepsilon, 11, 21, 22\}$ *dans le graphe* $\mathcal{G}_{\Lambda,L}$*, auquel on a ajouté la relation d'accessibilité de* Λ*.*

Pour montrer cette proposition, on commence d'abord par définir un codage d'un n-arbre de pile sur Γ comme un ensemble de n-piles sur $\Gamma \times \{\varepsilon, 11, 21, 22\}$ qui est l'ensemble des chemins de la racine aux feuilles où on ajoute la position de la feuille dans l'arbre (à l'aide des $11, 21, 22$). On définira ensuite trois types MSO-formules sur le treegraph d'ordre n de $\Gamma \times \{\varepsilon, 11, 21, 22\}$: $\delta(X)$ qui est satisfaite si X représente le codage d'un n-arbre de piles défini préalablement, et pour chaque automate normalisé d'opération A, les formules $\psi_A(X, Y)$ et $\phi_A(X, Y)$, la première étant satisfaite si il existe une opération D reconnue par A telle que le couple formé par le n-arbre de piles codé par X et celui codé par Y appartient à la relation définie par D, et la deuxième étant satisfaite si le couples de n-arbres de piles t_X et t_Y respectivement codés par X et Y est dans $\mathcal{R}_\mathcal{A}$. La normalisation de l'automate sera nécessaire pour que ces formules soient bien équivalentes à l'automate, comme cela sera précisé dans la preuve de la correction de ces formules. L'interprétation à ensembles finis qui permet de montrer la proposition précédente pour le graphe $\mathcal{G}_{R,L}$ sera donc la suivante :

$$((\delta(X) \wedge \exists Y, \psi_B([a_1]_n, Y) \wedge \phi_C(Y, X)), \psi_{A_{a_1}}(X, Y), \ldots, \psi_{A_{a_k}}(X, Y), \phi_C(X, Y))$$

où B est l'automate reconnaissant L à partir du n-arbre de piles à un seul nœud étiqueté par $[a_1]_{n-1}$, et C est l'automate reconnaissant la relation d'accessibilité de R.

L'interprétation à ensemble fini qui permet de montrer la proposition précédente pour le graphe $\mathcal{G}_{\mathcal{A}_{a_1},\cdots,\mathcal{A}_{a_k},L}$ sera la suivante :

$$((\delta(X) \wedge \exists Y, \psi_B([a_1]_n, Y) \wedge \phi_C(Y, X)), \phi_{A_{a_1}}(X, Y), \ldots, \phi_{A_{a_k}}(X, Y), \phi_C(X, Y))$$

où B est l'automate reconnaissant L à partir du n-arbre de piles à un seul nœud étiqueté par $[a_1]_{n-1}$, et C est l'automate reconnaissant la relation définie par $(\bigcup_{a\in\Sigma} \mathcal{A}_a)^*$.

Commençons donc par définir le codage d'un n-arbre de piles sur Γ comme un ensemble de n-piles sur $\Gamma \times \{\varepsilon, 11, 21, 22\}$. Pour plus de lisibilité, dans toute la suite de ce paragraphe, le couple (α, ε) sera simplement noté α.

Définition 4.13. *Étant donné un arbre de piles t et une position $u \in dom(t)$, on définit*

$$\mathrm{Code}(t,u) = [\mu_{t,u,1}(t(\varepsilon)), \mu_{t,u,2}(t(u_{\leq 1})), \dots, \mu_{t,u,|u|}(t(u_{\leq |u|-1})), t(u)]_n$$

où pour p une $(n-1)$-pile, t un n-arbre de pile, u un nœud de t et $i \leq |u|$, $\mu_{t,u,i}(p)$ est la pile p' telle que $\mathrm{rew}_{\alpha,(\alpha,\mathrm{ju_i})}(p,p')$ avec α est le plus haut élément de Γ de p, j est le nombre de fils du nœud $u_1 \cdots u_{i-1}$ de t, et u_i est la lettre suivante de u.

Tout arbre de piles t est alors encodé par l'ensemble fini de n-piles $X_t = \{\mathrm{Code}(t,u) \mid u \in fr(t)\}$, c.à.d l'ensemble des codes de ses feuilles.

Informellement, $\mathrm{Code}(t,u)$ est simplement la n-pile lue sur le chemin de la racine de t à u, dans laquelle à chaque $(n-1)$-pile représentant un nœud, on a ajouté son nombre de fils et le numéro du fils qui permet d'arriver à u.

Exemple 4.2. *Le code de l'arbre de pile t représenté sur la figure 4.1 est :*

$$X_t = \{ \begin{array}{l} [[[\alpha\alpha]_1[\beta\alpha(\beta,21)]_1]_2[[\alpha\alpha]_1[\alpha\alpha(\alpha,11)]_1]_2[[\alpha\beta]_1]_2]_3, \\ [[[\alpha\alpha]_1[\beta\alpha(\beta,22)]_1]_2[[\alpha\alpha]_1[\alpha]_1[(\beta,21)]_1]_2[[\beta\alpha]_1[\beta\alpha]_1[\beta]_1]_2]_3\ , \\ [[[\alpha\alpha]_1[\beta\alpha(\beta,22)]_1]_2[[\alpha\alpha]_1[\alpha]_1[(\beta,22)]_1]_2[[\alpha\beta\beta]_1[\alpha\beta]_1]_2]_3\} \end{array}$$

Avant d'expliciter les formules δ, ϕ_A et ψ_A, on définit quelques formules techniques qui nous faciliterons leur description.

4.4.1 Notations et formules techniques

On définit d'abord des formules sur $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$ qu'on utilisera pour construire l'ensemble des piles utilisées pour représenter des arbres de piles dans $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$.

Pour $\alpha,\beta \in \Gamma \times \{\varepsilon,11,21,22\}$, on définit $\psi_{\mathrm{rew}_{\alpha,\beta}}(x,y) = x \xrightarrow{\mathrm{rew}_{\alpha,\beta}} y$. Pour tout $i < n$, on définit $\psi_{\mathrm{copy}_i}(x,y) = x \xrightarrow{\#_i} y$ et $\psi_{\overline{\mathrm{copy}}_i}(x,y) = y \xrightarrow{\#_i} x$. Enfin, pour $i \in \{1,2\}$ et $d \leq i$, on définit $\psi_{\mathrm{copy}^i_n,d}(x,y) = \exists z_1,z_2, \bigvee_{\alpha\in\Gamma}(x \xrightarrow{\mathrm{rew}_{\alpha,(\alpha,\mathrm{id})}} z_1 \wedge z_1 \xrightarrow{\#_n} z_2 \wedge z_2 \xrightarrow{\mathrm{rew}_{(\alpha,\mathrm{id}),\alpha}} y)$.

Pour $\theta \in \mathrm{Ops}_{n-1}(\Gamma)$, $\psi_\theta(x,y)$ est vraie si y est obtenue en appliquant θ à x. $\psi_{\mathrm{copy}^i_n,d}(x,y)$ est vraie si y est obtenue en ajoutant id à la lettre la plus haute de x, puis en dupliquant sa $(n-1)$-pile la plus haute et finalement en retirant id de sa lettre la plus haute. Ces formules vont permettre de simuler l'application des opérations d'arbres de piles à des arbres de piles sur les éléments de leurs codes.

On donne maintenant une formule technique qui vérifie qu'une pile donnée y est obtenue depuis une pile x en utilisant uniquement les formules précédentes : $\mathrm{Reach}(x,y)$

$$\mathrm{Reach}(x,y) = \forall X, ((x \in X \wedge \forall z,z', (z \in X \wedge (\bigvee_{\theta\in\mathrm{Ops}_{n-1}\cup\mathbb{T}_{n-1}} \psi_\theta(z,z') \vee \bigvee_{i\in\{1,2\}}\bigvee_{d\leq i} \psi_{\mathrm{copy}^i_n,d}(z,z'))) \Rightarrow z' \in X) \Rightarrow y \in X)$$

Cette formule est vraie si pour tout ensemble de n-piles X, si x est dans X et que X est clos par les relations ψ_θ et $\psi_{\mathrm{copy}^i_n,d}$, alors y est dans X.

Lemme 4.24. *Pour toutes n-piles $x = [x_1,\cdots,x_m]_n$ et $y = [y_1,\cdots,y_{m'}]_n$, $\mathrm{Reach}(x,y)$ est vraie si et seulement si*

- *Pour tout $i < m, y_i = x_i$,*
- *Pour tout $i \in [m,m'-1]$, il existe $\theta \in \mathrm{Ops}_{n-1}(\Gamma)^*$, $\alpha \in \Gamma$ et $h \in \{11,21,22\}$ tels que $\mathrm{r}_{\mathrm{rew}_{\alpha,(\alpha,\mathrm{h})}} \circ \mathrm{r}_\theta(x_m,y_i)$,*
- *Il existe $\theta \in \mathrm{Ops}_{n-1}(\Gamma)^*$ tel que $\mathrm{r}_\theta(x_m,y_{m'})$.*

Démonstration. Supposons que $\mathrm{Reach}(x, y)$ soit vraie, alors on peut obtenir y à partir de x en lui appliquant une suite d'opérations de $\mathrm{Ops}_{n-1}(\Gamma)$ ou de la forme $\mathrm{rew}_{\alpha,(\alpha,\mathrm{h})}\mathrm{copy}_n\mathrm{rew}_{(\alpha,\mathrm{h}),\alpha}$, avec $h \in \{11, 21, 22\}$. On a donc bien y de la forme donnée dans le lemme.

Supposons que y soit de la forme donnée, alors y est obtenu à partir de x par une suite d'opération de $\mathrm{Ops}_{n-1}(\Gamma)$ ou de la forme $\mathrm{rew}_{\alpha,(\alpha,\mathrm{h})}\mathrm{copy}_n\mathrm{rew}_{(\alpha,\mathrm{h}),\alpha}$, avec $h \in \{11, 21, 22\}$. Ce qui montre bien que $\mathrm{Reach}(x, y)$ est vraie. □

Corollaire 4.25. *Pour toute n-pile x et $\alpha \in \Gamma$, $\mathrm{Reach}([\alpha]_n, x)$ est vraie si et seulement si il existe un arbre de piles t et un nœud u tel que $x = \mathrm{Code}(t, u)$.*

Démonstration. Supposons qu'il existe un arbre de piles t et un nœud u tel que $x = \mathrm{Code}(t, u)$. Alors

$$x = [\mu_{t,u,1}(t(\varepsilon)), \mu_{t,u,2}(t(u_{\leq 1})), \cdots, \mu_{t,u,|u|}(t(u_{\leq |u|-1})), t(u)]_n$$

Comme pour tout j, $t(u_{\leq j})$ est dans $\mathit{Stacks}_{n-1}(\Gamma)$, il existe un ρ_j dans $\mathrm{Ops}_{n-1}(\Gamma)^*$ tel que $\rho_j([\alpha]_{n-1}, t(u_{\leq j}))$. Donc, par le lemme précédent, $\mathrm{Reach}([\alpha]_n, x)$ est vraie.

Inversement, supposons que $\mathrm{Reach}([\alpha]_n, x)$ soit vraie. D'après le lemme 4.24, pour tout $j < |x|$, il existe $\theta \in \mathrm{Ops}_{n-1}(\Gamma)^*$, $\beta_j \in \Gamma$ et $h_j \in \{11, 21, 22\}$ tel que $\mathrm{r}_{\mathrm{rew}_{\beta_\mathrm{j},(\beta_\mathrm{j},\mathrm{h_j})}} \circ \mathrm{r}_\theta([\alpha]_n, x_j)$, et il existe $\theta \in \mathrm{Ops}_{n-1}(\Gamma)^*$ tel que $\mathrm{r}_\theta([\alpha]_n, x_{|x|})$.

Pour tout j, on considère que $h_j = i_j d_j$. On choisit un domaine d'arbre U tel que $d_1 \cdots d_{|x|-1} \in U$. On définit un arbre t de domaine U tel que pour tout j, $\mathrm{r}_{\mathrm{rew}_{\beta_\mathrm{j},(\beta_\mathrm{j},\mathrm{h_j})}}(t(d_1 \cdots d_{j-1}), x_j)$, tout nœud $d_1 \cdots d_j$ a i_j fils, et pour tout $u \in U$ qui n'est pas un $d_1 \cdots d_j$, $t(u) = [\alpha]_n$. On a alors $x = \mathrm{Code}(t, d_1 \cdots d_{|x|-1})$. □

4.4.2 La formule δ

On définit $\delta(X) = \mathrm{OnlyLeaves}(X)) \wedge \mathrm{TreeDom}(X) \wedge \mathrm{UniqueLabel}(X)$ avec

$$\begin{aligned}
\mathrm{OnlyLeaves}(X) &= \forall x, x \in X \Rightarrow \mathrm{Reach}([\alpha]_n, x)\\
\mathrm{TreeDom}(X) &= \forall x, y, z((x \in X \wedge \psi_{\mathrm{copy}^2_n,2}(y, z) \wedge \mathrm{Reach}(z, x)) \Rightarrow\\
&\quad \exists r, z'(r \in X \wedge \psi_{\mathrm{copy}^2_n,1}(y, z') \wedge \mathrm{Reach}(z', r)))\wedge\\
&\quad ((x \in X \wedge \psi_{\mathrm{copy}^2_n,1}(y, z) \wedge \mathrm{Reach}(z, x)) \Rightarrow\\
&\quad \exists r, z'(r \in X \wedge \psi_{\mathrm{copy}^2_n,2}(y, z') \wedge \mathrm{Reach}(z', r)))\\
\mathrm{UniqueLabel}(X) &= \forall x, y, (x \neq y \wedge x \in X \wedge y \in X) \Rightarrow\\
&\quad (\exists z, z', z'', \psi_{\mathrm{copy}^2_n,1}(z, z') \wedge \psi_{\mathrm{copy}^2_n,2}(z, z'')\wedge\\
&\quad ((\mathrm{Reach}(z', x) \wedge \mathrm{Reach}(z'', y)) \vee (\mathrm{Reach}(z'', x) \wedge \mathrm{Reach}(z', y))))
\end{aligned}$$

où α est une lettre de Γ.

La formule OnlyLeaves vérifie qu'un élément x de X code le nœud d'un arbre de pile. TreeDom vérifie que la clôture par préfixe de l'ensemble des mots u tels qu'il existe un arbre t, tel que $\mathrm{Code}(t, u) \in X$ est un domaine d'arbre, et que l'ensemble des mots $i_0 \cdots i_{m-1}$, où pour tout j, i_j est l'arité du nœud $u_1 \cdots u_j$ de t est inclus dans cet ensemble (en d'autres termes, que l'arité annoncée par les i_j est bien respectée). Enfin UniqueLabel vérifie que pour tout couple d'éléments $x = \mathrm{Code}(t, u)$ et $y = \mathrm{Code}(t', v)$ de X, il existe un indice $1 \leq j \leq \min(|u|, |v|)$ tel que pour tout $k < j$, $u_k = v_k$, $t(u_k) = t'(u_k)$ et le nœud u_k a la même arité dans t et t', et $u_j \neq v_j$, c'est-à-dire que pour tout couple d'éléments, les $(n-1)$-piles étiquetant les ancêtres communs sont égales, et que x et y ne peuvent coder la même feuille (étant donné que $u \neq v$). De plus, cela empêche également x de coder un nœud présent sur le chemin menant de la racine à y.

Lemme 4.26. *Pour tout X sous-ensemble fini de $\mathit{Stacks}_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$, $\delta(X) \iff \exists t \in ST_n, X = X_t$.*

Démonstration. On montre d'abord que pour tout n-arbre de piles t, $\delta(X_t)$ est vraie sur $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$. Par définition, pour tout $x \in X_t$, $\exists u \in fr(t), x = \text{Code}(t,u)$, et donc $\text{Reach}([\alpha]_n, x)$ est vraie (d'après le corollaire 4.25). Ainsi OnlyLeaves est vraie.

On prend $x \in X_t$ tel que $x = \text{Code}(t,u)$ avec $u = u_0 \cdots u_i 2 u_{i+2} \cdots u_{|u|}$. Comme t est un arbre, $u_0 \cdots u_i 2 \in dom(t)$ de même que $u_0 \cdots u_i 1$. Il existe donc $v \in fr(t)$ tel que $\forall j \leq i, v_j = u_j$, $v_{i+1} = 1$, et $\text{Code}(t,v) \in X_t$. Prenons maintenant $x \in X_t$ tel que $x = \text{Code}(t,u)$ avec $u = u_0 \cdots u_i 1 u_{i+2} \cdots u_{|u|}$ et le nœud $u_0 \cdots u_i$ a 2 fils dans t, alors $u_0 \cdots u_i 2$ est dans $dom(t)$ et il existe $v \in fr(t)$ tel que $\forall j \leq i, v_j = u_j$, $v_{i+1} = 2$ et $\text{Code}(t,v) \in X_t$. Ainsi TreeDom est vraie.

Soient x et y dans X_t tels que $x \neq y$, $x = \text{Code}(t,u)$ et $y = \text{Code}(t,v)$, et soit i le plus petit indice tel que $u_i \neq v_i$. Supposons que $u_i = 1$ et $v_i = 2$ (l'autre cas étant symétrique). On appelle $z = \text{Code}(t, u_0 \cdots u_{i-1})$, et on prend z' et z'' tels que $\psi_{\text{copy}^2_n,1}(z,z')$ et $\psi_{\text{copy}^2_n,2}(z,z'')$. On en déduit que $\text{Reach}(z',x)$ et $\text{Reach}(z'',y)$ sont vraies. Et par conséquent UniqueLabel est vraie. Ainsi, pour tout arbre de piles t, $\delta(X_t)$ est vraie.

Montrons maintenant que pour tout $X \subseteq \mathit{Stacks}_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$ tel que $\delta(X)$ est vraie, il existe $t \in ST_n$, tel que $X = X_t$. Comme OnlyLeaves est vraie, pour tous $x \in X$, il existe t_x un arbre de piles et u^x un nœud de t_x tels que $x = \text{Code}(t_x, u^x)$. On définit l'ensemble de mots $U = \{u \mid \exists x \in X, u \sqsubseteq u^x\}$. Par définition, U est clos par préfixe. Comme TreeDom est vraie, pour tout u, si $u2$ est dans U, alors $u1$ est aussi dans U. Ainsi U est bien un domaine d'arbre. De plus, s'il existe un x tel que $u1 \sqsubseteq u^x$ et le nœud u a 2 fils dans t_x, alors TreeDom assure qu'il existe un y tel que $u2 \sqsubseteq u^y$ et donc que $u2 \in U$. Comme UniqueLabel est vraie, pour tous x et y deux éléments distincts de X, il existe un j tel que pour tout $k < j$ on a $u^x_k = u^y_k$, et $u^x_j \neq u^y_j$. Alors, pour tout $k \leq j$, on a $x_k = y_k$ et donc le nœud $u^x_1 \cdots u^x_k$ a le même nombre de fils et la même étiquette dans t_x et dans t_y. Ainsi, pour tout $u \in U$, on peut définir σ_u tel que pour tout x tel que $u \sqsubseteq u^x$, $t_x(u) = \sigma_u$, et que le nombre de fils de chaque nœud soit cohérent avec le codage.

On considère l'arbre de pile t de domaine U tel que pour tout $u \in U$, $t(u) = \sigma_u$. On a $X = X_t$, ce qui conclue cette preuve. □

4.4.3 La formule ϕ_A associée à un automate normalisé A

Expliquons maintenant la formule $\phi_A(X,Y)$, qui peut être écrite comme

$$\exists Z_{q_1}, \cdots, Z_{q_{|Q|}}, \phi'_A(X,Y,\vec{Z})$$

avec

$$\phi'_A(X,Y,\vec{Z}) = \text{Init}(X,Y,\vec{Z}) \wedge \text{Diff}(\vec{Z}) \wedge \text{Trans}(\vec{Z})$$

Ces trois formules Init, Diff et Trans sont définies ci-après.

$$\text{Init}(X,Y,\vec{Z}) = (\bigcup_{q_i \in I} Z_{q_i}) \subseteq X \wedge (\bigcup_{q_i \in F} Z_{q_i}) \subseteq Y \wedge X \setminus (\bigcup_{q_i \in I} Z_{q_i}) = Y \setminus (\bigcup_{q_i \in F} Z_{q_i})$$

Cette formule vérifie que seules les feuilles de X sont étiquetées par des états initiaux, que seules les feuilles de Y sont étiquetées par des états finaux et qu'en dehors de leurs feuilles étiquetées, X et Y sont égaux (c.a.d. non modifiés).

$$\begin{aligned}\text{Diff}(\vec{Z}) = \big(\bigwedge_{q,q' \in Q_{T,c}} Z_q \cap Z_{q'} = \emptyset\big) \wedge \big(\bigwedge_{q,q' \in Q_{C,c}} Z_q \cap Z_{q'} = \emptyset\big) \\ \wedge \big(\bigwedge_{q,q' \in Q_{T,d}} Z_q \cap Z_{q'} = \emptyset\big) \wedge \big(\bigwedge_{q,q' \in Q_{C,d}} Z_q \cap Z_{q'} = \emptyset\big)\end{aligned}$$

Cette formule vérifie qu'une pile donnée (et donc une feuille donnée dans un arbre apparaissant dans un calcul) est étiquetée par au plus un état de chaque sous-ensemble de Q : $Q_{T,d}, Q_{C,d}, Q_{T,c}, Q_{C,c}$. Ainsi, si on doit faire un choix non déterministe lors du calcul, on ne pourra choisir qu'une possibilité.

$$\mathrm{Trans}(\vec{Z}) = \forall s, \bigwedge_{q\in Q} ((s \in Z_q) \Rightarrow (\bigvee_{K\in\Delta} \mathrm{Trans}_K(s,\vec{Z}) \vee \rho_q))$$

où ρ_q est vraie si et seulement si q est un état final, et

$$\begin{aligned}
\mathrm{Trans}_{(q,\mathrm{copy}^1_n,q')}(s,\vec{Z}) &= \exists t, \psi_{\mathrm{copy}^1_n,1}(s,t) \wedge t \in Z_{q'},\\
\mathrm{Trans}_{(q,\overline{\mathrm{copy}}^1_n,q')}(s,\vec{Z}) &= \exists t, \psi_{\mathrm{copy}^1_n,1}(t,s) \wedge t \in Z_{q'},\\
\mathrm{Trans}_{(q,\theta,q')}(s,\vec{Z}) &= \exists t, \psi_\theta(s,t) \wedge t \in Z_{q'}, \text{pour } \theta \in \mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1},\\
\mathrm{Trans}_{(q,(q',q''))}(s,\vec{Z}) &= \exists t,t', \psi_{\mathrm{copy}^2_n,1}(s,t) \wedge \psi_{\mathrm{copy}^2_n,2}(s,t') \wedge t \in Z_{q'} \wedge t' \in Z_{q''},\\
\mathrm{Trans}_{((q,q'),q'')}(s,\vec{Z}) &= \exists t,t', \psi_{\mathrm{copy}^2_n,1}(t',s) \wedge \psi_{\mathrm{copy}^2_n,2}(t',t) \wedge t \in Z_{q'} \wedge t' \in Z_{q''},\\
\mathrm{Trans}_{((q',q),q'')}(s,\vec{Z}) &= \exists t,t', \psi_{\mathrm{copy}^2_n,1}(t',t) \wedge \psi_{\mathrm{copy}^2_n,2}(t',s) \wedge t \in Z_{q'} \wedge t' \in Z_{q''}.
\end{aligned}$$

Cette formule vérifie que l'étiquetage respecte les règles de l'automate, et que pour toute pile étiquetée par q, si il existe une règle avec q dans sa partie gauche, il existe au moins une pile qui est le résultat de l'application d'une de ces règle à cette pile et qui est étiquetée par la partie droite de la règle. Et il est également possible pour un état final de ne pas avoir de successeur.

Proposition 4.27. *Étant donnés deux arbres de piles s et t, on a $\phi_A(s,t)$ vraie si et seulement si il existe des opérations $D_1, \cdots, D_k$ reconnues par A telles que t est obtenu en appliquant $D_1, \cdots, D_k$ à des positions disjointes de s.*

Démonstration. Supposons tout d'abord qu'il existe de tels $D_1, \cdots, D_k$. On va construire un étiquetage de $\mathit{Stacks}_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$ qui satisfasse $\phi_A(X_s, X_t)$. On choisit un étiquetage des D_i par A. On va étiqueter Stacks_n avec cet étiquetage. Si on obtient un arbre de piles t' à n'importe quel étape du calcul lors de l'application de D_i à s, on étiquette $\mathrm{Code}(t', u)$ par l'étiquette du nœud de D_i accrochée à la feuille à la position u de t'. Remarquons que cela ne dépend pas de l'ordre dans lequel on applique les D_i à s ni de par quelles feuilles on choisit d'appliquer les opérations en premier.

On suppose que $\mathrm{r}^{i_1}_{D_1} \circ \cdots \circ \mathrm{r}^{i_k}_{D_k}(s,t)$. Étant donné un nœud x de l'un des D_i, on appelle $l(x)$ son étiquetage.

Formellement, on définit l'étiquetage inductivement : le $(D_1, i_1, s_1), \cdots, (D_k, i_k, s_k)$-étiquetage de $\mathit{Stacks}_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$ est défini comme suit :

- Le $\emptyset$-étiquetage est l'étiquetage vide.
- Le $(D_1, i_1, s_1), \cdots, (D_k, i_k, s_k)$-étiquetage est l'union du (D_1, i_1, s_1)-étiquetage et du $(D_2, i_2, s_2), \cdots, (D_k, i_k, s_k)$-étiquetage.
- Le $(\Box, i, s)$-étiquetage est $\{\mathrm{Code}(s, u_i) \to l(x)\}$, où u_i est la $i^{\text{ème}}$ feuille de s et x est l'unique nœud de $\Box$.
- Le $(F_1 \cdot_{1,1} D_\theta) \cdot_{1,1} F_2, i, s)$-étiquetage est le $(F_1, i, s), (F_2, i, s')$-étiquetage, avec $\mathrm{r}^i_{F_1} \circ \mathrm{r}^i_\theta(s, s'))$.
- Le $((((F_1 \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} F_3) \cdot_{1,1} F_2), i, s)$-étiquetage est le (F_1, i, s), $(F_2, i, s'), (F_3, i+1, s')$-étiquetage, avec $\mathrm{r}^i_{F_1} \circ \mathrm{r}^i_{\mathrm{copy}^2_n}(s, s')$.
- Le $((F_1 \cdot_{1,1} (F_2 \cdot_{2,1} \overline{\mathrm{copy}}^2_n)) \cdot_{1,1} F_3, i, s)$-étiquetage est le $(F_1, i, s), (F_2, i + |I_{F_1}|, s)$, (F_3, i, s')-étiquetage, avec $\mathrm{r}^i_{F_1} \circ \mathrm{r}^{i+1}_{F_2} \circ \mathrm{r}^i_{\overline{\mathrm{copy}}^2_n}(s, s')$.

Observons que ce procédé termine. En effet, la somme du nombre d'arêtes et de nœuds des opérations diminue strictement à chaque pas.

On choisit $\vec{Z}$ le $(D_1, i_1, s), \cdots, (D_k, i_k, s)$-étiquetage de $\mathit{Stacks}_n(\Sigma \times \{\varepsilon, 11, 21, 22\})$.

Lemme 4.28. *L'étiquetage précédemment défini, $\vec{Z}$, satisfait $\phi'(X_s, X_t, \vec{Z})$.*

Démonstration. On commence par citer un lemme technique qui provient directement de la définition de l'étiquetage :

Lemme 4.29. *Étant donné une opération réduite D, un étiquetage de D, ρ_D, un arbre de pile t, un $i \in \mathbb{N}$ et un $j \leq |I_D|$, l'étiquette de* $\mathrm{Code}(t, u_{i+j-1})$ *(où u_i est la $i^{ème}$ feuille de t) dans le (D, i, t)-étiquetage est $\rho_D(x_j)$ (où x_j est le $j^{ème}$ nœud d'entrée de D).*

Par soucis de simplicité, considérons pour cette preuve que D est une opération réduite (si c'est un ensemble d'opération réduite, la preuve est la même pour chaque opération).

Tout d'abord, montrons que Init est satisfaite. D'après le lemme précédent, tous les nœuds de X_s sont étiquetées par les étiquettes des nœuds de D (ou non étiquetées), et sont donc étiquetés par des états initiaux (puisque nous avons considéré un étiquetage acceptant de D). De plus, comme l'automate est distingué, seuls ces nœuds peuvent être étiquetés par des états initiaux. De même, les nœuds de X_t et eux-seuls sont étiquetés par des états finaux (ou ils ne sont pas étiquetés).

On montre maintenant que Trans est satisfaite. Supposons que l'un des $\mathrm{Code}(t', u_i)$ est étiqueté par un état q. Par construction de l'étiquetage, on l'a obtenu via un $(\Box, i, t')$-étiquetage. Si q est final, il n'y a rien de plus à vérifier, étant donné que ρ_q est satisfaite. Si non, le nœud x étiqueté par q qui est le seul nœud de $\Box$ qui a étiqueté $\mathrm{Code}(t', u_i)$ par q a au moins un fils dans D. Supposons, par exemple que $D = (F_1 \cdot_{1,1} D_\theta) \cdot_{1,1} F_2$ tel que x est le nœud sortant de F_1. On appelle y le nœud d'entrée de F_2. Comme D est reconnue par A, il est étiqueté par un q' tel que $(q, \theta, q') \in \Delta_A$. Par construction, on prend le $(F_1, i, s), (F_2, i, t'')$-étiquetage, avec $\mathrm{r}^i_{F_2}(s, t')$ et $\mathrm{r}^i_\theta(t', t'')$. Ainsi, on a $\mathrm{Code}(t'', u_i)$ étiqueté par q' (d'après le lemme 4.29), et ainsi $\mathrm{Trans}_{(q,\theta,q')}(\mathrm{Code}(t', u_i), \vec{Z})$ est satisfaite, étant donné que $\psi_\theta(\mathrm{Code}(t', u_i), \mathrm{Code}(t'', u_i)$ l'est.

Les autres cas possibles pour décomposer D ($D = (((F_1 \cdot_{1,1} D_{\mathrm{copy}^1_n}) \cdot_{2,1} F_3) \cdot_{1,1} F_2$ ou $D = ((F_1 \cdot_{1,1} (F_2 \cdot_{2,1} \overline{\mathrm{copy}}^2_n)) \cdot_{1,1} F_3)$ sont similaires et sont de ce fait laissés au lecteur. Observons que D peut ne pas être décomposable depuis le nœud x, auquel cas on décompose D et on considère la partie contenant x jusqu'à ce qu'on puisse décomposer l'opération au nœud x, cas où l'argument est le même.

Montrons maintenant que l'étiquetage satisfait Diff. Étant donnés $q, q' \in Q_{C,d}$, supposons qu'il existe $\mathrm{Code}(t', u_i)$ étiquetés par q et q'. Par construction, cet étiquetage est obtenu par un $(F_1, i, t'_1), (F_2, i, t'_2)$-étiquetage, où F_1 et F_2 sont tous les deux $\Box$, et $t'_1(u_i) = t'_1(u_i)$. On appelle x (resp. y) l'unique nœud de F_1 (resp. F_2). x est étiqueté par q et y par q'.

Supposons que D peut être décomposé comme $(G \cdot_{1,1} D_\theta) \cdot_{1,1} H$ (ou $((G \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} K) \cdot_{1,1} H$, ou $(G \cdot_{1,1} (H \cdot_{1,2} D_{\overline{\mathrm{copy}}^2_n})) \cdot_{1,1} K$) tel que y est le nœud sortant de G (sinon, on décompose D jusqu'à pouvoir obtenir une telle décomposition). Ensuite, supposons qu'on peut décomposer comme $G = G_1 \cdot_{1,1} D_\theta \cdot_{1,1} G_2$ (ou $(G_1 \cdot_{1,1} (G_3 \cdot_{1,2} D_{\overline{\mathrm{copy}}^2_n})) \cdot_{1,1} G_2$. Comme on considère des états de $Q_{C,d}$, il n'y a pas d'autre cas possible) tel que x est le nœud d'entré de G_2. Ainsi, on a par construction $\mathrm{r}_{G_2}(\mathrm{Code}(t', u_i), \mathrm{Code}(t', u_i))$. Donc G_2 définit une relation contenue dans l'identité. Comme c'est une partie de D, et de ce fait étiqueté par des états de A, avec q et q' dans $Q_{C,d}$, il n'y pas de transitions étiquetées par copy^j_n ou $\overline{\mathrm{copy}}^j_n$ dans G_2. De plus, comme q et q' sont dans $Q_{C,d}$, G_2 n'est pas une opération contenant uniquement une transition de test. C'est donc une suite d'éléments de $\mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1}$ définissant une relation incluse dans l'identité. Comme A est normalisé, c'est impossible, et donc $\mathrm{Code}(t', u_i)$ ne peut pas être étiqueté à la fois par q et q'.

Si on prend deux états dans l'une des autres sous-parties de Q, on obtient la même contradictions avec peu de modifications, et on laisse donc ces cas au lecteur.

Ainsi, comme toutes ses sous-formules sont satisfaites, $\phi'_A(X_s, X_t, \vec{Z})$ est satisfaite avec l'étiquetage $\vec{Z}$. Et ainsi $\phi_A(X_s, X_t)$ est satisfaite. □

Supposons maintenant que $\phi_A(X_s, X_t)$ est satisfaite. On prend un étiquetage minimal $\vec{Z}$ qui satisfait la formule $\phi'_A(X_s, X_t, \vec{Z})$. On construit le graphe suivant D :

$$\begin{array}{rcl}
V_D & = & \{(x,q) \mid x \in \mathit{Stacks}_n(\Gamma \times \{\varepsilon, 11, 21, 22\}) \wedge x \in Z_q\} \\
E_D & = & \{((x,q), \theta, (y,q')) \mid (\exists \theta, (q,\theta,q') \in \Delta \wedge \psi_\theta(x,y))\} \\
 & \cup & \{((x,q), 1, (y,q')), ((x,q), 2, (z,q'')) \mid (q,(q',q'')) \in \Delta \\
 & & \wedge \psi_{\mathrm{copy}^2_n,1}(x,y) \wedge \psi_{\mathrm{copy}^2_n,2}(x,z)\} \\
 & \cup & \{((x,q), \bar{1}, (z,q'')), ((y,q'), \bar{2}, (z,q'')) \mid ((q,q'),q'') \in \Delta \\
 & & \wedge \psi_{\mathrm{copy}^2_n,1}(z,x) \wedge \psi_{\mathrm{copy}^2_n,2}(z,y)\} \\
 & \cup & \{((x,q), 1, (y,q')) \mid (q, \mathrm{copy}^1_n, q') \in \Delta \wedge \psi_{\mathrm{copy}^1_n,1}(x,y)\} \\
 & \cup & \{((x,q), \bar{1}, (y,q')) \mid (q, \overline{\mathrm{copy}}^1_n, q') \in \Delta \wedge \psi_{\mathrm{copy}^1_n,1}(y,x)\}
\end{array}$$

Lemme 4.30. *D est une union disjointe d'opérations $D_1, \cdots, D_k$.*

Démonstration. Supposons que D n'est pas un DAG. Il existe alors $(x,q) \in V$ tel que $(x,q) \xrightarrow{+} (x,q)$, et donc il existe une suite d'opérations de A_d (pour A_c le cas est symétrique, et il n'existe aucune transition de A_c vers A_d, ainsi, un état ne peut pas avoir d'état dans les deux sous-parties) qui est l'identité (et donc une suite d'opérations de $\mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1}$). Étant donné que A_d est normalisé, il est impossible d'avoir une telle suite. Ainsi, D ne contient aucun circuit et est donc un DAG.

Par définition de E_D, il est étiqueté par $\mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1} \cup \{1, \bar{1}, 2, \bar{2}\}$.

On choisit l'un des D_i. Supposons que ce n'est pas une opération. Ainsi, il existe un nœud (x,q) de D_i tel que D_i ne peut être décomposé en ce nœud (c'est-à-dire, que dans la décomposition inductive, aucun cas ne peut être appliqué pour pour décomposer D_i ou l'une de ces sous-opération pour obtenir (x,q) comme nœud sortant (ou d'entrée) de la sous-opération obtenue. On considère les cas suivant pour le voisinage de (x,q) :

- (x,q) a un unique fils (y,q'), qui n'a pas d'autre père tel que $(x,q) \xrightarrow{2} (y,q')$. Par définition de Trans, on a $\psi_{\mathrm{copy}^2_n,2}(x,y)$, et ainsi on a $(q,(q'',q')) \in \Delta$ et il existe un z dans $Z_{q''}$ tel que $\psi_{\mathrm{copy}^2_n,1}(x,z)$. Cela contredit que (x,q) a un seul fils dans D_i. Si $(x,q) \xrightarrow{\bar{2}} (y,q')$, le cas est similaire. Pour tout autre $\theta \in \mathrm{Ops}_{n-1} \cup \mathbb{T}_{n-1} \cup \{1, \bar{1}\}$, on peut décomposer le sous-DAG $\{(x,q) \xrightarrow{\theta} (y,q')\}$ en $(\Box \cdot_{1,1} D_\theta) \cdot_{1,1} \Box$.
- Supposons que (x,q) a au moins trois fils $(y_1,q_1), (y_2,q_2), (y_3,q_3)$. Il n'y a aucune sous-formule de Trans qui impose d'étiqueter trois nœuds qui peuvent être obtenus à partir de x, ce qui contredit la minimalité de l'étiquetage.

  Pour les mêmes raisons, (x,q) a au plus deux pères.
- Supposons que (x,q) possède deux fils (y_1,q_1) et (y_2,q_2). Par définition de Trans et par minimalité de l'étiquetage, on a $\psi_{\mathrm{copy}^2_n,1}(x,y_1)$, $\psi_{\mathrm{copy}^2_n,2}(x,y_2)$, et $(q,(q_1,q_2\,)) \in \Delta$ (autrement, l'étiquetage ne serait pas minimal, étant donné que c'est la seule sous-formule imposant d'étiqueter deux fils d'un nœud). Ainsi, on a $(x,q) \xrightarrow{1} (y_1,q_1)$ et $(x,q) \xrightarrow{2} (y_2,q_2)$. Toujours par minimalité, (y_1,q_1) et (y_2,q_2) n'ont pas d'autre père que (x,q). Dans ce cas, le sous-DAG $\{(x,q) \xrightarrow{1} (y_1,q_1), (x,q) \xrightarrow{2} (y_2,q_2)\}$ peut être décomposé comme $((\Box \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} \Box) \cdot_{1,1} \Box$.
- Supposons que (x,q) a un seul fils (y_1,q_1) qui a un autre père (y_2,q_2). Par définition de Trans et par minimalité de l'étiquetage, on a que $\psi_{\mathrm{copy}^2_n,1}(y_1,x)$, $\psi_{\mathrm{copy}^2_n,2}(y_1,y_2)$, et $((q,q_2),q_1) \in \Delta$. On a ainsi $(x,q) \xrightarrow{\bar{1}} (y_1,q_1)$ et $(y_2,q_2) \xrightarrow{\bar{2}} (y_1,q_1)$. Par minimalité encore, (y_2,q_2) n'a pas d'autre fils que (y_1,q_1). Dans ce cas, le sous-DAG $\{(x,q) \xrightarrow{\bar{1}} (y_1,q_1), (y_2,q_2) \xrightarrow{\bar{2}} (y_1,q_1)\}$ peut être décomposé en $(\Box \cdot_{1,1} (\Box \cdot_{1,2} D_{\overline{\mathrm{copy}}^2_n})) \cdot_{1,1} \Box$.

Dans tous les cas considérés, ou le cas est impossible, ou le DAG est décomposable au nœud (x,q). Ainsi le DAG D_i est toujours décomposable et est donc ainsi une opération.

□

Lemme 4.31. *Tous les D_i sont reconnus par A*

Démonstration. Par construction, pour tout nœud (x, q), si $x \in X_s$, q est un état initial (car init est satisfaite), et (x, q) est donc un nœud d'entrée car A est distingué. Et comme init est satisfaite, seuls ces nœuds sont étiquetés par des états initiaux.

De plus, pour tout nœud (x, q), si $x \in X_t$, q est un état final (car init est satisfaite) et (x, q) est alors un nœud sortant, car A est distingué. Et comme init est satisfaite, seuls ces nœuds sont étiquetés par des états finaux.

Par construction, les arêtes sont toutes des transitions de Δ, et on étiquette donc chaque nœud (x, q) par q.

Étant donné que la formule Trans est satisfaite, on a que pour tout nœud (x, q), soit q est final (et donc (x, q) est un nœud sortant), soit on a l'un des cas suivant :

- un nœud (y, q') et θ tel que $\psi_\theta(x, y)$ et $(q, \theta, q') \in \Delta$
- deux nœud (y, q') et (z, q'') tels que $\psi_{\mathrm{copy}^2_n,1}(x, y)$, $\psi_{\mathrm{copy}^2_n,2}(x, z)$ et $(q, (q', q'')) \in \Delta$
- deux nœuds (y, q') et (z, q'') tels que $\psi_{\mathrm{copy}^2_n,1}(z, x)$, $\psi_{\mathrm{copy}^2_n,2}(z, y)$ et $((q, q'), q'') \in \Delta$

Alors, seuls les nœuds (x, q) avec q final n'ont pas de fils et sont ceux étiquetés par des états finaux. De même, seuls les nœuds (x, q) avec q initial n'ont pas de père.

Ainsi, tout D_i est reconnu par A avec cet étiquetage. □

Lemme 4.32. *t est obtenu en appliquant les D_i à des positions disjointes de s.*

Démonstration. On montre par induction que $\mathrm{r}^j_D(s, t')$ si et seulement si $X_{t'} = X_s \cup \{x \mid (x, q) \in O_D\} \backslash \{x \mid (x, q) \in I_D\}$:

- Si $D = \square$, la propriété est vérifiée car $X_{t'} = X_s$ et $t' = s$.
- Si $D = (F \cdot_{1,1} D_\theta) \cdot_{1,1} G$, par hypothèse d'induction, on considère r tel que $\mathrm{r}^j_F(s, r)$, on a alors $X_r = X_s \cup \{y\} \backslash \{x \mid (x, q) \in I_F\}$, où (y, q') est le seul nœud sortant de F. Par construction, le nœud d'entrée de G, (z, q'') est tel que $\psi_\theta(y, z)$, et on a donc $\mathrm{r}^j_\theta(r, r')$ tel que $X_{r'} = X_r \backslash \{y\} \cup \{z\}$. Par hypothèse d'induction, on a $X_{t'} = X_{r'} \cup \{x \mid (x, q) \in O_G\} \backslash \{z\}$, car $\mathrm{r}^j_F \circ \mathrm{r}^j_\theta \circ \mathrm{r}^j_G(s, t')$ et donc $\mathrm{r}^j_G(r', t')$. Ainsi, $X_{t'} = X_s \cup \{x \mid (x, q) \in O_G\} \backslash \{x \mid (x, q) \in I_F\} = X_s \cup \{x \mid (x, q) \in O_D\} \backslash \{x \mid (x, q) \in I_D\}$.

Les autres cas étant similaires, ils sont laissés au lecteur. Il suffit alors de construire successivement t_1 et t_2 tels que $\mathrm{r}^{i_1}_{D_1}(s, t_1)$, $\mathrm{r}^{i_2}_{D_2}(t_1, t_2)$, etc, pour obtenir t et prouver le lemme.

□

On a ainsi prouvé les deux directions : pour tous n-arbres de piles s et t, il existe un ensemble d'opérations D_i reconnues par A tels que t est obtenus en appliquant les D_i à des positions disjointes de s si et seulement si $\phi_A(X_s, X_t)$ est satisfaite. □

4.4.4 La formule ψ_A associée à un automate normalisé A

La formule ψ_A est vraie si les arbres définis par ses deux variables sont reliés par une seule opération reconnue par A. De ce fait, elle ne diffère de ϕ_A que par ce point, ϕ_A reliant des arbres reliés par un nombre arbitraire d'opérations reconnues par A, appliquées à des positions disjointes. On la définit ainsi :

$$\psi_A(X, Y) = \exists Z_{q_1}, \cdots, Z_{q_{|Q|}}, \phi'_A(X, Y, \vec{Z}) \wedge \mathrm{Connected}(X, Y, \vec{Z})$$

avec

$$\mathrm{Connected}(X, Y, \vec{Z}) = \exists z, \bigvee_{q \in Q} z \in Z_q \wedge \forall x, ((x \in X \vee x \in Y) \wedge \bigvee_{q \in Q} x \in Z_q) \Rightarrow \mathrm{Reach}(z, x)$$

Informellement, le sommet z représente un point d'articulation de l'opération reconnue, c'est à dire accessible depuis tous les sommets entrants et depuis lequel tous les sommets sortants sont accessibles. Cela se traduit par le fait que la n-pile codant le nœud de l'arbre obtenu à ce sommet peut mener à tous les sommets de X et de Y étiquetés par la formule via des opérations basique en ne passant que par des sommets étiquetés. Et les autres feuilles ne peuvent pas être étiquetées.

Proposition 4.33. *Étant donnés deux arbres de piles s et t, on a $\psi_A(X_s, X_t)$ vraie si et seulement si il existe une opération D reconnue par A telle que $\mathrm{r}_D(s,t)$.*

Démonstration. Étant donné que ϕ_A est contenue dans ψ_A, on a de même que précédemment que $\psi_A(X_s, X_t)$ si et seulement si il existe $D_1, \cdots, D_k$ reconnues par A telles que t est obtenu en appliquant les D_i à s de manière disjointe. Pour obtenir notre propriété, il suffit de montrer que ψ_A est vérifiée si et seulement si on a de plus $k = 1$.

Supposons tout d'abord qu'il existe un D reconnu par A tel que $\mathrm{r}^i_D(s,t)$. On définit le même étiquetage qu'au paragraphe précédent, et le lemme 4.28 est vérifié. Il suffit d'y ajouter le lemme suivant pour conclure que $\psi_A(X_s, X_t)$ est vérifiée :

Lemme 4.34. *L'étiquetage satisfait* $\mathrm{Conneted}(X_s, X_t, \vec{Z})$.

Démonstration. On choisit un point d'articulation de D, c'est à dire un sommet x tel que pour tout sommet entrant i de D, $i \xrightarrow[D]{*} x$ et pour tout sommet sortant f de D, $x \xrightarrow[D]{*} f$. On demande de plus qu'aucun arc étiqueté par $\bar{1}$ ne soit présent sur un descendant de x, et aucun arc étiqueté par 1 ne soit présent sur un ancêtre de x. On choisit un étiquetage de D, et on suppose que x est étiqueté par l'état q.

Par définition du (D, i, s)-étiquetage de $\mathit{Stacks}_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$, si $D = D_1 \cdot_{1,1} D_2$, avec x le nœud sortant de D_1 et le nœud entrant de D_2, et si t' est l'arbre tel que $\mathrm{r}^i_{D_1}(s,t')$ et $\mathrm{r}^i_{D_2}(t',t)$, on a $\mathrm{Code}(t', u_i)$ étiqueté par q.

Étant donné que D_1 ne contient aucune opération copy^d_n, on a que pour tout feuille de s qui n'est pas une feuille de t', on a une opération θ de $\mathrm{Ops}_n(\Gamma \times \{\varepsilon, 11, 21, 22\})^*$ où n'apparaissent que des opérations de $\mathrm{Ops}_{n-1}(\Gamma)$ ou $\mathrm{rew}_{\alpha,(\alpha,\mathrm{h})}\overline{\mathrm{copy}}_n\mathrm{rew}_{(\alpha,\mathrm{h}),\alpha}$, avec $h \in \{11, 21, 22\}$, telle que $\mathrm{r}^i_\theta(\mathrm{Code}(t', u_i), \mathrm{Code}(s,u))$. On a donc $\mathrm{Reach}(\mathrm{Code}(t', u_i), \mathrm{Code}(s,u))$. De la même manière, c'est également le cas pour les feuilles de t qui ne sont pas des feuilles de t', ce qui conclue le lemme. □

Supposons maintenant que $\psi_A(X_s, X_t)$ soit vraie. De même que précédemment, on construit le graphe D à partir d'un étiquetage minimal $\vec{Z}$ satisfaisant la formule. On a les lemmes 4.30, 4.31 et 4.32. Il suffit de montrer qu'on a $k = 1$ dans le lemme 4.30. Étant donné que $\mathrm{Conneted}(X_s, X_t, \vec{Z})$ est satisfaite, il existe un sommet z étiqueté par un état q tel que pour toute feuille u de s étiquetée par un état, on a $\mathrm{Reach}(z, \mathrm{Code}(s,u))$, et pour toute feuille u de t étiquetée par un état, on a $\mathrm{Reach}(z, \mathrm{Code}(t,u))$.

De même que dans le lemme 4.30 puisque z est étiqueté par q et que l'étiquetage est minimal, il existe un chemin de chaque $(\mathrm{Code}(s,u), q_i)$ à (z,q) et un chemin de (z,q) à chaque $(\mathrm{Code}(t,u), q_f)$, et donc que D est un unique DAG connexe, et donc est une seule opération, ce qui conclue la démonstration.

□

Ces trois formules nous permettent de définir les interprétations à ensembles finis annoncés au début du paragraphe, ce qui permet bien de démontrer le théorème 4.22.

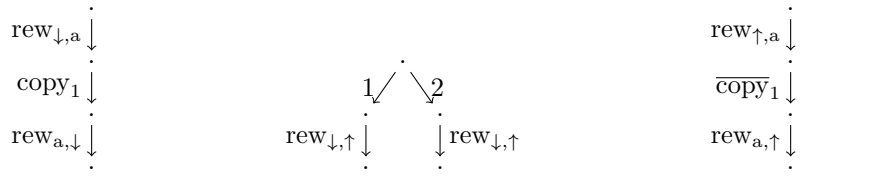
4.5 Traces des graphes de réécritures d'arbres de piles

On peut considérer les systèmes de réécriture suffixes d'arbres de piles comme des accepteurs de langages. Pour cela, on considère un alphabet Γ, un *ensemble de départ* $T_i \subseteq ST_n$ fini ou

reconnaissable, un *ensemble d'arrivée* $T_f \subseteq ST_n$ fini ou reconnaissable, et pour un système de réécriture suffixe d'arbres de piles R étiqueté par un alphabet Σ. Un mot $a_1 \cdots a_k$ est reconnu à partir de T_i si il existe $t_0, \cdots, t_k \in ST_n$ tels que $t_0 \in T_i$, $t_k \in T_f$ et pour tout $i < k$, $t_i \xrightarrow[R]{a_i} t_{i+1}$ (pour la version sans ε pour la version avec ε, il faut ajouter un nombre arbitraire de transitions étiquetées par ε autour de chaque lettre).

On peut définir de manière similaire les langages définis par des transducteurs suffixes d'arbres de piles.

Exemple 4.3. *On reprend le système de l'exemple 4.1 : on fixe un alphabet Σ et deux symboles spéciaux $\uparrow$ et $\downarrow$. on considère $ST_2(\Sigma \cup \{\uparrow, \downarrow\})$. On définit le système de réécriture R, dont les règles sont données en figure 4.11, avec, pour tout $a \in \Sigma$, $R_a = \{D_a\}$, et $R_\varepsilon = \{\mathrm{Dupl}\} \cup \{P_a \mid a \in \Sigma\}$.*



(a) P_a, pour chaque $a \in \Sigma$ (b) *Dupl* (c) D_a, for every $a \in \Sigma$

FIGURE 4.11 – Les règles du système de réécriture

Les ensembles de départ et d'arrivée sont représentés en figure 4.12.

(a) L'arbre de départ. (b) Un arbre d'arrivée. s est une 1-pile arbitraire.

FIGURE 4.12 – Les arbres de départ et d'arrivée.

Le langage des traces reconnu par ce système est

$$\{P_{a_1} \cdots P_{a_n} \cdot Dupl \cdot ((D_{a_n} \cdots D_{a_1}) \sqcup\!\sqcup (D_{a_n} \cdots D_{a_1})) \mid a_1, \cdots, a_n \in \Sigma\}.$$

Expliquons informellement pourquoi. On commence par l'arbre de piles initial, qui possède un seul nœud étiqueté par une 1-pile dont le symbole le plus haut est $\downarrow$ (et qui contient uniquement cette lettre). On ne peut donc pas lui appliquer D_a. Si on lui applique P_a, on reste dans la même situation, en ayant ajouté un a juste en dessous de $\downarrow$. On peut donc lire une suite d'opérations $P_{a_1} \cdots P_{a_k}$. Depuis cet état, on peut également appliquer Dupl, qui produit un arbre de piles avec trois nœuds, dont les deux feuilles sont étiquetées par $[a_1 \cdots a_k \uparrow]_1$ (si on a au préalable lu $P_{a_1} \cdots P_{a_n}$). À partir de cet arbre de piles, on peut seulement appliquer des règles D_a. Si les deux feuilles sont étiquetées respectivement par $[b_1 \cdots b_m \uparrow]_1$ et $[c_1 \cdots c_\ell \uparrow]_1$, on peut appliquer soit D_{b_m} ou D_{c_ℓ}, donnant l'arbre de piles obtenu à partir du précédent en retirant b_m de la feuille de gauche ou c_ℓ de la feuille de droite. On peut appliquer une opération D_a jusqu'à ce qu'il ne reste un arbre de piles dont les deux feuilles sont étiquetées par $[\uparrow]_1$ qui est un arbre final. Ainsi, sur chaque feuille, on va lire $D_{a_n} \cdots D_{a_1}$ dans cet ordre, mais on n'a aucune contrainte sur l'alternance entre ces deux suites. On va donc bien pouvoir obtenir toutes les suites de $(D_{a_n} \cdots D_{a_1}) \sqcup\!\sqcup (D_{a_n} \cdots D_{a_1})$. Et seules ces suites permettent d'accéder à un arbre final.

D'après ce qui précède, un mot d'opération qui permet d'aller de l'arbre initial est

$$w \in \bigcup_{a_1 \cdots a_n \in \Sigma^*} P_{a_1} \cdots P_{a_k} Dupl(D_{a_n} \cdots D_{a_1}) \sqcup\!\sqcup (D_{a_n} \cdots D_{a_1})$$

et donc l'étiquette de ce chemin est $\lambda(w)$ appartient à $\bigcup_{a_1 \cdots a_n \in \Sigma^*} (a_n \cdots a_1) \sqcup\!\sqcup (a_n \cdots a_1) = \{u \sqcup\!\sqcup u \mid u \in \Sigma^*\}$. Pour conclure, il suffit juste de remarquer que l'ensemble des arbres final est bien un ensemble d'arbres de piles reconnaissables (grâce à l'automate reconnaissant R^*), et on a bien montré que ce langage est reconnaissable par un système de réécriture d'arbres de piles.

On ne connaît pas exactement la classe de langage reconnue par les systèmes de réécriture d'arbres de piles, qu'on notera $\mathcal{L}_{GSTRS}$. Cependant, on peut citer quelques propriétés faciles sur cette classe de langages.

La proposition 3.20 est également vérifiées par les systèmes de réécriture suffixes d'arbres de piles.

Proposition 4.35. *La classe $\mathcal{L}_{\mathrm{GSTRS}_n}$ des langages reconnus par les systèmes de réécriture suffixe d'arbres de piles d'ordre n contient strictement les langages algébriques.*

$\mathcal{L}_{\mathrm{GSTRS}_n}$ est close par union.

Le premier point étant vrai à l'ordre 1, et les systèmes d'ordre n pouvant toujours être vus comme des systèmes d'ordre $n+1$, ils est trivialement vrai à tous les ordres. Le deuxième point provient simplement du fait que l'union de deux systèmes de réécriture suffixe d'arbres de piles est un système de réécriture suffixe d'arbres de piles.

On a également les propriétés faciles suivantes.

Proposition 4.36. *La classe de langages reconnue par des systèmes de réécriture de piles d'ordre n est contenue dans $\mathcal{L}_{\mathrm{GSTRS}_n}$.*

Étant donné que tout automate à pile peut être considéré comme un système de réécriture d'arbres de piles, l'inclusion est évidente.

Proposition 4.37. $\mathcal{L}_{\mathrm{GTRS}} \subseteq \bigcup_n \mathcal{L}_{\mathrm{GSTRS}_n}$.

Un système de réécriture suffixe d'arbres est un système de réécriture suffixe d'arbres de piles d'ordre 1, donc l'inclusion est évidente.

Proposition 4.38. *$\mathcal{L}_{\mathrm{GSTRS}_n}$ est incluse dans la classe des langages reconnus par les graphes arbre-automatiques d'ordre n.*

Les graphes des systèmes de réécriture suffixes d'arbres de piles étant, grâce à la démonstration de la décidabilité de $FO[\rightarrow^*]$ sur ceux-ci, arbre-automatiques d'ordre supérieur, cette propriété est évidente.

Notons que dans les trois propriétés précédente, nous n'avons pas cité d'inclusion stricte. Nous conjecturons que c'est le cas, mais n'en avons pour l'instant pas de preuve formelle. Nous pensons cependant que l'exemple 4.3 fourni un séparateur pour les deux premières propositions. Pour la troisième, l'absence de toute autre description des graphes arbre-automatique d'ordre supérieur que celle par interprétation à ensemble fini ne nous permet pas de déterminer un tel séparateur pour l'instant.

Conjecture 1. *La classe de langages reconnue par des systèmes de réécriture de piles d'ordre n est strictement contenue dans $\mathcal{L}_{\mathrm{GSTRS}_n}$.*

Si nous n'avons pas de démonstration formelle, nous pouvons cependant présenter une intuition de pourquoi $\{u \sqcup\!\sqcup u \mid u \in \Sigma^*\}$ ne peut pas être reconnu par un automate à piles d'ordre supérieur. Pour reconnaître ce langage, il faut être capable de déterminer un mot u et de lire parallèlement deux copies de u de manière indépendante. La seule manière de générer le mot est

de stocker son miroir dans une 1-pile. On peut facilement obtenir une 2-pile contenant deux 1-piles contenant chacune le miroir de u. On veut ensuite commencer à lire $u \sqcup\!\sqcup u$. Tant qu'on veut lire des lettres de la première copie de u, il suffit de dépiler sur la 1-pile supérieure. Cependant à tout moment, on doit pouvoir basculer sur la deuxième copie de u, tout en conservant l'état de la pile de niveau supérieur. Si on élimine simplement la 1-pile la plus haute, on va perdre cette information. La seule solution pour faire ceci est donc d'avoir une 3-pile qui contient cette 2-pile, de la copier et d'ensuite éliminer la 1-pile supérieure et de basculer sur la deuxième copie, à partir de laquelle on commence à lire des lettres de la deuxième copie. Si on veut maintenant rebasculer sur la première copie de u, il faut à nouveau conserver l'information de l'état de la deuxième copie de u. Or cette information n'est présente que dans la 2-pile la plus haute, alors que la première copie n'est présente que dans la 2-pile la plus basse. On ne peut donc pas se permettre de dépiler simplement. Il faut donc être dans une 4-pile et copier la 3-pile, puis dépiler la 2-pile, etc...

En itérant ce raisonnement, on s'aperçoit que si on effectue n basculements entre les deux copies de u, on a besoin d'avoir un automate à pile d'ordre $n+1$. Cependant, étant donné un mot u, il peut être nécessaire d'effectuer au plus $2|u|$ basculements entre les deux copies, si on veut lire le mot $u_1u_1u_2u_2\cdots u_{|u|}u_{|u|}$. La taille du mot u pouvant être arbitrairement grande, on a besoin d'un automate à pile de niveau infini pour pouvoir reconnaître ce langage. Or un automate à pile d'ordre supérieur a forcément un niveau fini. Et donc ce langage n'est pas reconnu par un automate à pile d'ordre supérieur.

Conjecture 2. $\mathcal{L}_{\text{GTRS}} \subsetneq \bigcup_n \mathcal{L}_{\text{GSTRS}_n}$.

À nouveau, nous conjecturons que le langage $\{u \sqcup\!\sqcup u \mid u \in \Sigma^*\}$ est un séparateur de ces deux classes. Expliquons informellement pourquoi. Dans les système de réécriture d'arbre, il n'y a pas d'opération de copie d'un mot. Or, pour reconnaître ce langage, il faut être capable de générer deux copies d'un même mot qu'on détruit ensuite de manière disjointe. Un système de réécriture suffixe d'arbres est incapable de réaliser des opérations de copie d'un mot entier. Une solution serait de contraindre l'application des même opérations à deux sous-arbres disjoint, ce qui est impossible.

On conjecture de plus que la hiérarchie formée par les classes de langages reconnues par les systèmes de réécriture suffixes d'ordre n forment une hiérarchie stricte.

Conjecture 3. *Pour tout* $n \geq 0$, $\mathcal{L}_{\text{GSTRS}_n} \subsetneq \bigcup_n \mathcal{L}_{\text{GSTRS}_{n+1}}$.

L'inclusion est évidente. Pour montrer qu'elle est stricte, il suffirait de trouver un langage reconnus par un système d'ordre $n+1$ qui ne peut pas être reconnu au niveau n. On peut conjecturer qu'un langage reconnu par un automate à pile d'ordre n ne peut être reconnu par un système de réécriture suffixe d'arbres de piles d'ordre $n-1$. De plus, on a déjà conjecturé que $\{u \sqcup\!\sqcup u \mid u \in \Sigma^*\}$ est un séparateur entre les langages reconnus à l'ordre 1 et les autres ordres. Nous n'avons cependant pour l'instant pas trouvé d'exemple simple candidat à être un séparateur à tout ordre.

Chapitre 5

Traces de systèmes à compteurs

Dans ce chapitre, nous nous intéressons aux graphes générés par des systèmes à compteurs, et notamment à leur langages de traces. Jusqu'ici, nous avons considéré une approche «par en-dessous», consistant à prendre des classes de graphes existantes et à les généraliser en tentant de conserver la décidabilité du plus de propriétés possibles. Nous considérons maintenant une approche «par au-dessus», c'est-à-dire que nous partons d'un modèle très général, les systèmes à compteurs, qui est Turing-complet, et on va chercher à déterminer des manière de l'affaiblir de manière à récupérer la décidabilité de certaines propriétés. Nous nous intéresserons notamment à la décidabilité de l'algébricité d'un langage, qui est un problème indécidable en général, et qui, dans les modèles que nous définirons sera décidable. On considérera deux restrictions des systèmes à compteurs : les systèmes d'addition de vecteurs et les systèmes à compteurs plats.

Dans un premier temps, nous définissons la notion de système à compteurs, et rappelons que c'est un formalisme Turing-complet. Nous présentons ensuite les systèmes d'addition de vecteurs qui en sont une restriction connue, et rappelons un état de l'art des problèmes de décidabilité connus sur ce modèle. Nous rappelons ensuite la caractérisation de l'algébricité d'un langage dit borné due à Ginsburg et Spanier à l'aide d'ensembles semi-linéaire, et rappelons le problème de stratifiabilité d'un ensemble semi-linéaire dont la décidabilité est inconnue à ce jour. Nous montrons que dans le cas des polyèdres d'entiers, ce problème est décidable et coNP-complet. Nous appliquons ensuite ce résultat pour montrer que si un système à compteurs est plat (c'est-à-dire que ses traces sont incluses dans un langage borné reconnaissable), on peut décider l'algébricité de son langage de traces et que ce problème est coNP-complet. Enfin, on donne une nouvelle preuve de l'algébricité du langage des traces d'un système d'addition de vecteurs, problème déjà étudié dans [Sch92].

5.1 Systèmes à compteurs

Définition 5.1. *Un* système à compteurs de dimension c *est un couple* $\mathcal{S} = \langle \Theta, \vec{c}_{\text{Init}} \rangle$ *avec*

- $\Theta \subseteq \bigcup_{m \geq 0} \mathbb{Z}^{m \times c} \times \mathbb{Z}^m \times \mathbb{Z}^c$ *un ensemble fini de transitions,*
- $\vec{c}_{init} \in \mathbb{N}^c$ *un vecteur initial.*

Une configuration *de* $\mathcal{S}$ *est un vecteur de* $\mathbb{N}^c$.

Étant données deux configurations $\vec{x}, \vec{y}$ *et une transition* $\theta = (\mathbf{A}, \vec{b}, \vec{v})$, *on a* $\vec{x} \xrightarrow{\theta} \vec{y}$ *si*

- $\mathbf{A}\vec{x} \geq \vec{b}$
- $\vec{y} = \vec{x} + \vec{v}$.

Une *trace* d'un système à compteurs est une suite $w \in \Theta^*$ telles qu'il existe des configurations $\vec{x}_0, \cdots, \vec{x}_{|w|}$ avec $\vec{x}_0 = \vec{c}_{\text{Init}}$ et pour tout $i < |w|$, on a $\vec{x}_i \xrightarrow{w_i} \vec{x}_{i+1}$.

Le graphe induit par un système à compteurs est $\mathcal{G}_\mathcal{S} = (V, E)$, avec :

- $V = \mathbb{N}^c$.
- $E = \{(\vec{v}, \theta, \vec{v'}) \mid \vec{v} \xrightarrow{\theta} \vec{v'}\}$.

On considérera les graphes enracinés en $\vec{c}_{\text{Init}}$ des systèmes à compteurs, qui sont définis par

- $V = \{\vec{v} \in \mathbb{N}^c \mid \exists \tau \in \Theta^*, \vec{c}_{\text{Init}} \xrightarrow{\tau} \vec{v}\}$,
- $E = \{(\vec{v}, \theta, \vec{v'}) \mid \vec{v} \xrightarrow{\theta} \vec{v'}\}$.

On peut également remarquer qu'au lieu d'avoir un étiquetage de chaque règle, on utilise directement la règle elle-même pour étiqueter chaque arc. Cela est dû au fait que si l'on considérait un étiquetage quelconque de ces arcs, la plupart des problèmes (notamment ceux sur les langages de traces) que nous considérerons sur ces modèles sont indécidables. Le langage des traces de $\mathcal{S}$ est le langage $\mathcal{L}(\mathcal{S}) = \{\tau \mid \exists \vec{v}, \vec{c}_{\text{Init}} \xrightarrow{\tau} \vec{v}\}$.

Les systèmes à compteurs sont un formalisme assez ancien et possédant de nombreuses caractérisations équivalentes. Nous en avons donné un formalisme autorisant des transitions assez expressives, mais il existe des formalismes autorisant à produire des graphes isomorphes avec des transitions beaucoup plus basiques. L'un de ces formalismes les plus connus est celui dit des machines de Minsky. Dans ce modèle, les configurations sont constituées d'un état et d'un ensemble de compteurs, et les seules transitions existantes permettent d'incrémenter un compteur, d'en décrémenter un, ou de tester si un compteur est égal à 0. On peut également parler des machines à deux compteurs, qui constituent un autre formalisme équivalent à celui que nous avons présenté. Tous ces modèles sont équivalents aux machines de Turing, comme le montre Minsky dans [Min67]. De ce fait, tous les problèmes indécidables sur les machines de Turing le sont également sur les systèmes à compteurs, comme par exemple, déterminer si une configuration donnée est accessible, déterminer si le langage reconnu par un système à compteurs est vide, l'égalité des langages de deux systèmes à compteurs, ou la régularité ou l'algébricité d'un tel langage, pour ce qui nous intéresse ici. Aussi, nous nous intéresserons à des restrictions de ces modèles sur lesquels nous montrerons que ce dernier problème est décidable : les systèmes d'addition de vecteurs (que nous présentons ci-après), et les systèmes à compteurs plats, qui sont l'intersection d'un système à compteurs avec un automate reconnaissant un langage de la forme $\sigma_1^* \cdots \sigma_k^*$.

5.2 Systèmes d'addition de vecteurs

Les systèmes d'addition de vecteurs, introduits dans [KM69] sont un modèle qui jouit d'une grande popularité et qui a été largement étudié à partir des années 1970, de par leur équivalence avec les réseaux de Petri, qui sont très utilisés pour modéliser des systèmes réels. Nous résumons ainsi les principaux résultats de décidabilité connus sur les systèmes d'addition de vecteurs, après en avoir donné la définition – dans le formalisme que nous avons choisi.

Définition 5.2. *Un* système d'addition de vecteurs de dimension c *est un système à compteurs de dimension c tel que pour toute transition $(\mathbf{A}, \vec{b}, \vec{v}) \in \Theta$, $\mathbf{A} = \mathbf{0}$ et $\vec{b} = \vec{0}$. On considère dans ce cas que Θ est un sous-ensemble fini de $\mathbb{Z}^c$.*

Problème de couverture Le problème de couverture revient à se demander si il est possible de *couvrir* un vecteur donné, c'est-à-dire d'atteindre un vecteur par le système d'addition de vecteur qui lui est strictement supérieur. En général, ce problème est lié au calcul de l'ensemble de couverture, contenant tous les vecteurs couverts par le système, qui donne une enveloppe convexe de l'ensemble des vecteurs accessibles. Ce problème est :

Entrée : Un système d'addition de vecteurs $\mathcal{S} = (\Theta, \vec{c}_{\text{Init}})$ et un vecteur $\vec{v}$.

Sortie : Si il existe un vecteur $\vec{v'} \geq \vec{v}$ tel que $\vec{c}_{\text{Init}} \underset{\mathcal{S}}{\rightarrow} \vec{v'}$.

Ce problème a été étudié par Karp et Miller dans [KM69]. Pour le résoudre, ils introduisent la construction d'un arbre, appelé depuis *arbre de Karp et Miller*, qui permet de calculer entièrement l'ensemble de couverture du système d'addition de vecteurs, et qui est toujours fini, prouvant de ce fait que l'ensemble de couverture (défini comme l'ensemble des vecteurs tel qu'il existe un vecteur plus grand accessible par le système d'addition de vecteurs) est bien calculable. Cet arbre est construit de la manière suivante : sa racine est étiqueté par la configuration initiale. Ensuite, tant qu'il existe une feuille non traitée, on lui ajoute comme fils toutes les configurations obtenues en lui appliquant une transitions, modifiées de la manière suivante :

- si une configuration est présente sur l'un des ancêtres de cette feuille, on ne l'ajoute pas,
- si une configuration est strictement supérieure à une configuration présente sur l'un des ancêtres de cette feuille, on remplace chacune des composantes strictement augmentées par ω, pour représenter que cette composante peut prendre une valeur arbitrairement grande,
- sinon, on l'ajoute telle quelle.

Ainsi, pour toute configuration apparaissant dans cet arbre, en remplaçant les ω par n'importe quelle valeur finie, il existe une configuration strictement supérieure accessible dans le système d'addition de vecteurs.

Cet arbre peut également être utilisé pour déterminer si l'ensemble d'accessibilité est fini. En effet, il est fini si et seulement si ω n'apparaît pas dans l'arbre, l'ensemble des configurations apparaissant dans l'arbre étant dans ce cas l'ensemble d'accessibilité du système.

Cependant, si cet arbre est toujours fini, il existe des cas où sa taille est non élémentaire en fonction de la taille du système d'addition de vecteurs. La complexité de ce problème est donc a priori non-élémentaire. En fait, il a plus tard été montré que cette complexité est EXPSpace-complète [CLM76, Rac78].

Cette construction est centrale dans la plupart des études faites sur les systèmes d'addition de vecteurs, notamment les propriétés présentées ci-après. Elles utilisent en général la notion de *graphe de couverture*, qui est simplement l'arbre de Karp et Miller dans lequel on ajoute un arc étiqueté par une action $\vec{a}$ entre deux sommets $\vec{x}$ et $\vec{y}$ si $\vec{y} = \vec{x} + \vec{a}$ et où on confond les sommets ayant la même étiquette. Alternativement, si une feuille de l'arbre produit une étiquette déjà existante dans l'arbre, on ajoute un arc entre cette feuille et l'autre sommet étiqueté par l'action.

Exemple 5.1. *On considère le système d'addition de vecteurs de dimension* 2, $\mathcal{S} = (\{\vec{a} = (2,-1), \vec{b} = (-1,2)\}, (1,1))$. *On représente son arbre de Karp et Miller à la figure 5.1, et son graphe de couverture à la figure 5.2. On peut observer que puisque la configuration* (ω,ω) *apparaît dans l'arbre, tout vecteur de dimension* 2 *est couvert par* $\mathcal{S}$.

Exemple 5.2. *On présente un autre exemple, celui donné par Karp et Miller dans [KM69] où la notion est introduite. On considère le système d'addition de vecteurs de dimension* 5, $\mathcal{S} = (\{\vec{a} = (-1,1,0,0,0), \vec{b} = (-1,0,0,1,0), \vec{c} = (0,-1,2,0,0), \vec{d} = (0,1,-1,0,0), \vec{e} = (0,0,0,-1,2), \vec{f} = (0,0,0,1,-1)\}, (1,0,0,0,0))$.

On représente son arbre de Karp et Miller à la figure 5.3. On peut exhiber de nombreux vecteurs qui ne sont pas couverts par ce système d'addition de vecteurs, comme par exemple, le vecteur $(1,1,1,1,1)$.

Problème d'accessibilité Le problème d'accessibilité consiste à se demander s'il est possible d'atteindre un vecteur donné dans un système d'addition de vecteur. Le problème d'accessibilité est le suivant :

Entrée : Un système d'addition de vecteurs $\mathcal{S} = (\Theta, \vec{c}_{\text{Init}})$ et un vecteur $\vec{v}$.

Sortie : Si $\vec{c}_{\text{Init}} \underset{\mathcal{S}}{\longrightarrow} \vec{v}$.

La décidabilité de ce problème a été étudiée dans un premier temps par Tenney et Sacerdote dans [ST77], puis prouvée par Mayr dans [May81], qui en donne la première preuve correcte. Une preuve plus simple a ensuite été fournie par Kosaraju dans [Kos82], puis à nouveau simplifiée par

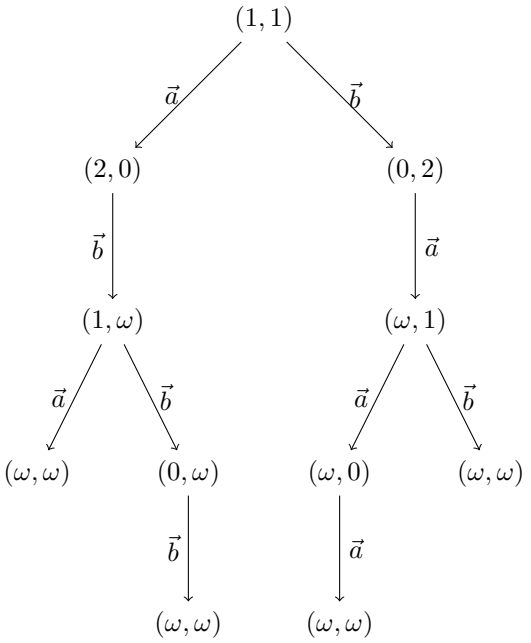


FIGURE 5.1 – L'arbre de Karp et Miller de l'exemple 5.1

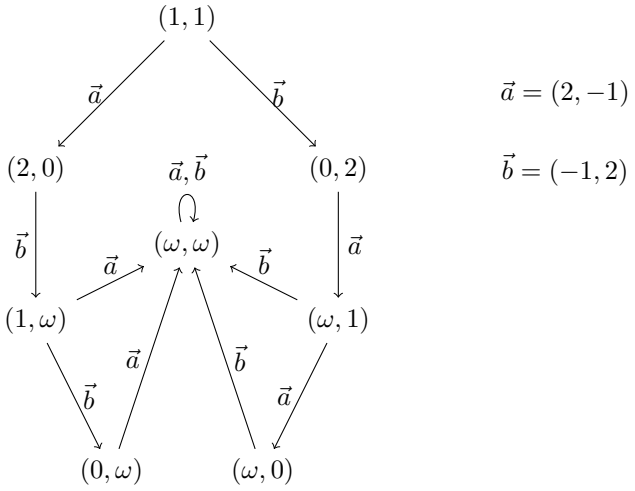


FIGURE 5.2 – Le graphe de couverture de l'exemple 5.1

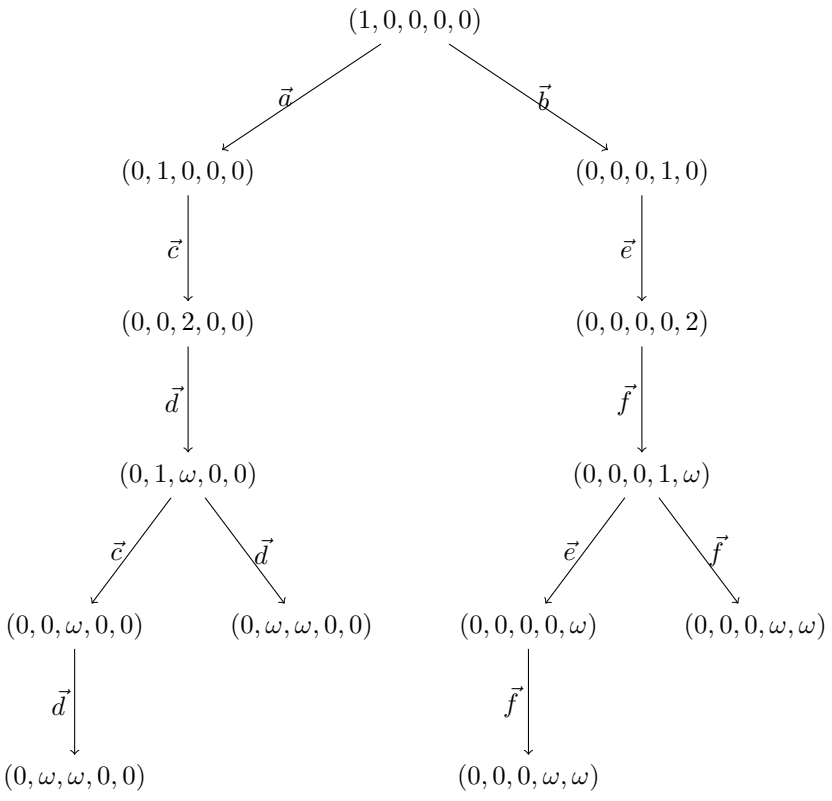


FIGURE 5.3 – L'arbre de Karp et Miller de l'exemple 5.2

Lambert dans [Lam92], toutes ces preuves utilisant une construction similaire appelée décomposition de Kosaraju-Lambert-Mayr-Sacerdote-Tenney, qui donne a priori une construction de complexité non-élémentaire. Par la suite, Leroux a donné dans [Ler10] un algorithme décidant ce même problème qui ne se base plus sur cette construction, mais sur la satisfaction de formules de l'arithmétique de Pressburger, et l'intersection de deux ensembles pseudo-linéaires. Par ailleurs, cet algorithme n'a pas a priori une complexité non-élémentaire. La complexité de ce problème est inconnue à ce jour. Cardoza, Lipton et Meyer ont montré dans [CLM76] que ce problème est EXPSpace-dur. Aucune borne supérieure pour ce problème n'est à ce jour connue.

Par ailleurs, l'ensemble des configurations accessibles n'est pas calculable. Dans [KM69], Karp et Miller attribuent ce résultat à Rabin, sous la forme du problème suivant :

Entrée : Deux systèmes d'addition de vecteurs $\mathcal{S}$ et $\mathcal{S}'$.

Sortie : $\mathcal{S}$ et $\mathcal{S}'$ ont le même ensemble de vecteurs accessibles.

Ce problème est indécidable.

Enfin, dans [Ler13c], Leroux a étudié le problème de l'accessibilité réversible pour les systèmes d'addition de vecteurs, qui est le suivant :

Entrée : Un système d'addition de vecteurs $\mathcal{S} = (\Theta, \vec{c}_{\text{Init}})$ et deux vecteurs $\vec{v_1}$ et $\vec{v_2}$.

Sortie : Si $\vec{v_1} \underset{\mathcal{S}}{\rightarrow} \vec{v_2}$, et $\vec{v_2} \underset{\mathcal{S}}{\rightarrow} \vec{v_1}$.

Leroux prouve également l'EXPSpace-complétude de ce problème.

Problème de la régularité des traces Le problème de la régularité des traces est le suivant :

Entrée : Un système d'addition de vecteurs $\mathcal{S} = (\Theta, \vec{c}_{\text{Init}})$ et un vecteur $\vec{v}$.

Sortie : Si le langage des traces $\mathcal{L}(\mathcal{S})$ est un langage reconnaissable.

La décidabilité de ce problème a été montrée de manière indépendante (et simultanée), par Ginzburg et Yoeli dans [GY80] d'une part et par Valk et Vidal-Naquet dans [VVN81] d'autre part. Les deux preuves utilisent l'arbre de Karp et Miller comme base, et le critère de reconnaissabilité du langage d'un système d'addition de vecteurs peut être résumé ainsi : le langage est reconnaissable si aucune composante ne peut être décrémentée un nombre arbitraire de fois. Dit, autrement, le critère est : «pour toute suite de transition $\tau_1 \cdots \tau_k$, si elle peut être répétée un nombre arbitraire de fois, alors elle ne diminue aucune composante».

La complexité de ce problème possède également une complexité EXPSpace-complète.

Problème de l'algébricité des traces Le problème d'algébricité des traces est le suivant :

Entrée : Un système d'addition de vecteurs $\mathcal{S} = (\Theta, \vec{c}_{\text{Init}})$ et un vecteur $\vec{v}$.

Sortie : Si le langage des traces $\mathcal{L}(\mathcal{S})$ est un langage algébrique.

Dans [Sch92], Schwer s'attaque au problème de l'algébricité des traces d'un système d'addition de vecteurs. Sa preuve repose sur la construction d'un graphe de couverture défini à partir de l'arbre de Karp et Miller, en ajoutant les arcs retour si une configuration d'un ancêtre devrait apparaître comme fils d'une feuille, puis en dépliant les circuits ne pouvant être empruntés qu'un nombre fini de fois (dit «faibles»). Elle étudie ensuite les circuits restants (dits «forts», ou «itérables»), et dégage cinq propriétés nécessaires et suffisantes pour l'algébricité du langage des traces. Malheureusement, [Sch92] contient un certains nombre d'inexactitudes, rendant la preuve qui y est présentée invérifiable. Par ailleurs, comme elle repose sur la construction effective du graphe de couverture, la complexité du processus de décision est au mieux non-élémentaire.

Leroux, Sutre et l'auteur ont repris ce problème dans [LPS13a], avec une preuve qui n'est plus basée directement sur la construction du graphe de couverture, et ont ensuite étudié ce problème dans le cas des systèmes à compteurs plats dans [LPS14]. Ces preuves sont inspirées des travaux de Ginsburg sur les langages bornés et les ensembles semi-linéaires (que nous rappelons dans le paragraphe suivant), déjà présents dans les travaux de Schwer. Le reste du chapitre est consacré à la présentation de ces résultats, en donnant dans un premier temps les résultats obtenus dans

[LPS14], puis ceux de [LPS13a]. Notons aussi que Leroux, Praveen et Sutre ont montré dans [LPS13b] que la complexité du problème d'algébricité des systèmes d'addition de vecteurs est EXPSPACE-complet.

5.3 Caractérisation des images de Parikh de langages algébriques bornés

Les langages bornés ont été introduits par Ginsburg et Spanier dans [GS64] comme un outil pour l'étude des langages algébriques. Formellement, un langage L est borné si il existe une suite finie de mots $\sigma_1, \ldots, \sigma_k$ telle que $L \subseteq \sigma_1^* \cdots \sigma_k^*$. Les langages bornés permettent de donner un critère de non algébricité assez simple : si l'intersection d'un langage avec un langage de la forme $\sigma_1^* \cdots \sigma_k^*$ n'est pas algébrique, alors le langage d'origine n'est pas algébrique. Cependant, dans le cas général, la réciproque est fausse, comme le montre l'exemple du langage $\{u_1 \cdots u_k \mid k \geq 1 \wedge u_i = a^i b^i\}$, qui n'est pas algébrique, mais dont l'intersection avec tout langage de la forme $\sigma_1^* \cdots \sigma_k^*$ est finie. Leur utilisation est notamment centrale dans [Sch92]. Ils occuperont également une place centrale dans notre preuve de l'algébricité des systèmes d'addition de vecteurs, et on peut noter que dans ce cas, le langage des traces d'un système d'addition de vecteurs est algébrique si et seulement si son intersection avec tout langage de la forme $\sigma_1^* \cdots \sigma_k^*$ est algébrique. Ils permettent également dans le cas de modèles Turing-complets, de se restreindre à des modèles dits «plats» qui possèdent en général des ensembles d'accessibilité calculables, via des techniques d'accélération ([ABS01],[BFLP08],[BW94],[BH99],[BIK10],[CJ98],[FIS03]), et qui permettent donc de mieux comprendre le comportement des systèmes Turing-complets dont ils sont issus. [EGM12] présente une manière d'unifier les récentes techniques de «vérification bornée».

Dans [GS64] où ils sont introduits, Ginsburg donne un critère nécessaire et suffisant de l'algébricité d'un langage borné, à partir des travaux de Parikh sur les langages algébriques ([Par61], [Par66]) : un langage borné est algébrique si et seulement si son image de Parikh est un ensemble semi-linéaire stratifié. Cependant, la stratifiabilité d'un ensemble semi-linéaire est à ce jour toujours ouverte. Ce paragraphe résume ces résultats.

Dans [Par61], Parikh introduit la notion d'ensembles linéaires et semi-linéaires.

Définition 5.3. *Un sous-ensemble L de $\mathbb{N}^d$ est dit* linéaire *si*

$$L = \vec{b} + \mathbb{N}\vec{p_1} + \cdots \mathbb{N}\vec{p_n}$$

où $\vec{b} \in \mathbb{N}^d$ est la base *de L et $P = \{\vec{p_1}, \cdots, \vec{p_n}\} \subseteq \mathbb{N}^d$ est l'ensemble de ses* périodes

P est dit stratifié si chacun de ses vecteurs a au plus deux composantes non nulles et si pour tous $1 \leq i < j < k < l \leq d$, il n'existe pas deux vecteurs $\vec{p_a}$ et $\vec{p_b}$ de P tel que $\vec{p_a}(i) > 0$, $\vec{p_a}(k) > 0$, $\vec{p_b}(j) > 0$ et $\vec{p_b}(l) > 0$

Étant donné un ensemble fini de vecteurs $P = \{\vec{p_1}, \cdots, \vec{p_k}\}$, on notera P^* l'ensemble linéaire canonique associé à P, $P^* = \mathbb{N}\vec{p_1} + \cdots + \mathbb{N}\vec{p_k}$.

Définition 5.4. *Un sous-ensemble S de $\mathbb{N}^d$ est dit* semi-linéaire *si c'est une union finie d'ensembles linéaires.*

Il est dit semi-linéaire stratifiable si c'est une union finie d'ensembles linéaires dont les périodes sont stratifiées.

Dans [GS66], Ginsburg et Spanier montrent que les ensembles semi-linéaires sont exactement les ensembles de $\mathbb{N}^c$ qui satisfont une formule de la logique de Presburger, c'est-à-dire la logique du premier ordre sur $\mathbb{N}$, avec des formules atomiques de la forme $t_0 + \sum_{i=1}^{n} t_i x_i = t'_0 + \sum_{i=1}^{n'} t'_i x_i$, où les t_i, t'_i sont des entiers et les x_i sont des variables. Cette logique a été introduite par Presburger dans [Pre30], et il a également montré que cette logique est décidable. Pour ce

qui nous concerne, cette caractérisation en terme de modèles de formules de Presburger des ensembles semi-linéaires a été exploitée pour étudier des propriétés des systèmes d'addition de vecteurs par Leroux, par exemple dans [Ler10], [Ler13c], [Ler13b], [Ler13a].

Lemme 5.1. *La classe des ensembles semi-linéaires stratifiables est close par union, projection, projection inverse, et par intersection avec des produits cartésiens d'intervalles.*

Démonstration. Ces propriétés s'obtiennent facilement à partir de la définition.

On considère deux ensembles semi-linéaires stratifiables $S = \bigcup_{i\leq n} \vec{b}_i + P_i^*$ et $S' = \bigcup_{i\leq n'} \vec{b'}_i + P_i'^*$.

Leur union est un ensemble semi-linéaire par définition : $S \cup S' = \bigcup_{i\leq n+n'} \vec{b''}_i P_i''^*$ avec $\vec{b''_i} = \vec{b}_i$ si $i \leq n$ et $\vec{b''}_i = \vec{b'}_{i-n}$ si $n < i \leq n+n'$; et $P''_i = P_i$ si $i \leq n$ et $P''_i = P'_{i-n}$ si $n < i \leq n+n'$. Toutes les périodes étant stratifiées, l'ensemble obtenu est stratifiable.

Soit π une projection sur un certain nombre de coordonnées. On montre que $\pi(S) = \bigcup_{i\leq n} \pi(\vec{b_i}) + \mathbb{N}\pi(\vec{p}_{i,1}) + \cdots + \mathbb{N}\pi(\vec{p}_{i,n_i})$: soit k une composante conservée par π et $\vec{v} \in S$, on a $\vec{v}(k) = \vec{b}_i(k) + \lambda_1\vec{p}_{i,1}(k) + \cdots \lambda_{n_i}\vec{p}_{i,n_i}(k)$, pour un certain i. On a $\pi(\vec{v})(k) = \pi(\vec{b_i})(k) + \lambda_1\pi(\vec{p}_{i,1})(k) + \cdots \lambda_{n_i}\pi(\vec{p}_{i,n_i})(k)$. Ainsi, on a bien $\pi(\vec{v}) = \pi(\vec{b_i}) + \lambda_1\pi(\vec{p}_{i,1}) + \cdots \lambda_{n_i}\pi(\vec{p}_{i,n_i})$, et donc $\pi(S) \subseteq \bigcup_{i\leq n} \pi(\vec{b_i}) + \mathbb{N}\pi(\vec{p}_{i,1}) + \cdots + \mathbb{N}\pi(\vec{p}_{i,n_i})$. Inversement, si on prend un vecteur $\vec{v'} = \pi(\vec{b_i}) + \lambda_1\pi(\vec{p}_{i,1}) + \cdots \lambda_{n_i}\pi(\vec{p}_{i,n_i})$ pour un certain i, on a que $\vec{v} = \vec{b}_i + \lambda_1\vec{p}_{i,1} + \cdots \lambda_{n_i}\vec{p}_{i,n_i}$ est un vecteur de S tel que $\pi(\vec{v}) = \vec{v'}$. Et donc on a bien l'égalité voulue, qui montre bien que $\pi(S)$ est un ensemble semi-linéaire. De plus, comme les P_i sont stratifiés, leur projection l'est aussi puisqu'on a simplement annulé certaines composantes.

La projection inverse revient à rajouter toutes les valeurs possibles pour les coordonnées ajoutées. Supposons que π est la projection qui élimine la coordonnée k (si elle en élimine plusieurs, le cas est similaire, il suffit de composer plusieurs projections éliminant une seule coordonnée). On considère donc l'ensemble semilinéaire $S'' = \bigcup_{i\leq n} \vec{b}_i + (P_i \cup \{\vec{e}_k\})^*$, où $\vec{e}_k$ est le vecteur valant 1 sur la coordonnée k et 0 sur les autres. On a facilement que $\pi(S'') = S$. Soit $\vec{v} \in S$. On a $\vec{v} = \vec{b}_i + \lambda_1\vec{p}_{i,1} + \cdots \lambda_{n_i}\vec{p}_{i,n_i}$ pour un certain i. On prend $\vec{v'} \in \pi^{-1}(\vec{v})$. Pour tout $k' \neq k$, on a $\vec{v'}(k') = \vec{v}(k')$. Ainsi, on a $\vec{v'} = \vec{b}_i + \lambda_1\vec{p}_{i,1} + \cdots \lambda_{n_i}\vec{p}_{i,n_i} + \vec{v'}(k)\vec{e}_k$. Ainsi $\vec{v'} \in S''$. Et donc $S'' = \pi^{-1}(S)$, qui est bien semilinéaire stratifiable puisqu'on a uniquement ajouté le vecteur e_k qui a une seule composante non nulle et que P_i est stratifié.

On prend un produit cartésien d'intervalles $\mathcal{I} = I_1 \times \cdots \times I_n$. Observons que $\mathcal{I} = I_1 \times \mathbb{N} \times \cdots \times \mathbb{N} \cap \cdots \cap \mathbb{N} \times \cdots \times \mathbb{N} \times I_n$, ce qui est une intersection de produits cartésiens d'intervalles, $\mathbb{N}$ en étant un. On va donc montrer la propriété pour $\mathcal{I} = I_1 \times \mathbb{N} \times \cdots \times \mathbb{N}$, et pour un ensemble linéaire $L = \vec{b} + \mathbb{N}\vec{p}_1 + \cdots \mathbb{N}\vec{p}_l$, puisque l'union est distributive par rapport à l'intersection, c'est suffisant. On a deux cas à traiter : $I_1 = [a, b]$ et $I_1 = [a, +\infty[$.

Commençons par $I_1 = [a, b]$. Dans ce cas, on prend comme ensemble de périodes $P' = \{\vec{p}_i \mid \vec{p}_i(1) = 0\}$, et comme ensemble de vecteurs de base $B = \{\vec{v} = \vec{b} + \lambda_1\vec{p}_1 + \cdots \lambda_n\vec{p}_n \mid \vec{p}_i \in P' \Rightarrow \lambda_i = 0 \wedge \vec{v}(1) \in [a, b]\}$. L'ensemble B est fini puisqu'on ajoute au vecteur de base uniquement des vecteurs qui ont une valeur non nulle sur leur première composante, et que cette composante doit être inférieure à b, on n'a donc qu'un nombre fini de telles sommes. Par définition, on a bien $L' = B + P'^* \subseteq L \cap \mathcal{I}$. Soit $\vec{v} \in L \cap \mathcal{I}$. On a $\vec{v} = \vec{b} + \lambda_1\vec{p}_1 + \cdots \lambda_n\vec{p}_n$ avec $\vec{v}(1) \in [a, b]$. On a donc que $\vec{b} + \lambda'_1\vec{p}_1 + \cdots \lambda'_n\vec{p}_n$ avec $\lambda'_i = \lambda_i$ si $\vec{p}_i(1) > 0$ et $\lambda_i = 0$ sinon, est un élément de B. Les autres $\vec{p}_i$ étant des éléments de P', on a bien $\vec{v} \in B + P'^*$. Et donc $L \cap \mathcal{I} = B + P'^*$, et est donc bien semi-linéaire.

Dans le cas où $I_1 = [a, +\infty[$, on prend $B = \{\vec{v} \in L \mid \vec{v} \geq a \wedge \forall \vec{p}_i, \vec{v} - \vec{p}_i \notin L \vee (\vec{v} - \vec{p}_i)(1) < a\}$. Cet ensemble est fini. En effet, si il était infini, on aurait deux éléments $\vec{b} + \lambda_1\vec{p}_1 + \cdots \lambda_n\vec{p}_n$ et $\vec{b} + \lambda'_1\vec{p}_1 + \cdots \lambda'_n\vec{p}_n$ tels que $\lambda'_i > \lambda_i$ pour tout i, ce qui est une contradiction avec sa définition. On a facilement que $L \cap [a, +\infty[= B + P^*$, simplement en remarquant que tout élément de B est dans L et que chaque élément de $L \cap [a, +\infty[$ est plus grand qu'un élément de B, par définition de B.

Ainsi, $L \cap \mathcal{I}$ est un ensemble semi-linéaire. Dans chaque cas, l'ensemble des périodes est conservé ou est remplacé par un de ses sous-ensembles. Comme il est stratifié, le nouvel ensemble des périodes l'est également, ce qui conclut la démonstration. □

Les ensembles semi-linéaires sont utilisés pour caractériser les langages algébriques. Dans [Par66], Parikh donne une caractérisation nécessaire pour qu'un langage soit algébrique :

Théorème 5.2. *Si un langage $L \subset \Sigma^*$ est algébrique, alors l'ensemble de vecteurs $\{(|u|_{a_1}, \cdots, |u|_{a_k}) \mid u \in L\}$ est un ensemble semi-linéaire (avec $\Sigma = \{a_1, \cdots, a_k\}$, l'ordre des lettres de Σ étant indifférent).*

L'ensemble $\{(|u|_{a_1}, \cdots, |u|_{a_k}) \mid u \in L\}$ est appelé *image de Parikh* du langage L.

Exemple 5.3. *Le langage $\{a^n \mid n$ est premier$\}$ n'est pas un langage algébrique. Son image de Parikh est l'ensemble des nombres premiers, qui n'est pas un ensemble semi-linéaire.*

Exemple 5.4. *Montrons un exemple où cette condition n'est pas suffisante, c'est-à-dire un langage L dont l'image de Parikh est un ensemble semi-linéaire, mais qui n'est pas algébrique.*

On considère $\Sigma = \{a, b, c, d\}$ et $L = \{a^n b^m c^n d^m \mid n, m \geq 0\}$. Ce langage n'est pas algébrique. Son image de Parikh est $\{(n, m, n, m) \mid n, m \geq 0\}$ et peut s'exprimer comme $(0, 0, 0, 0) + \mathbb{N}(1, 0, 1, 0) + \mathbb{N}(0, 1, 0, 1)$, et est de ce fait un ensemble semi-linéaire.

Remarque 5.1. *Le fait que cet exemple montre que la condition n'est pas suffisante provient du fait que le langage $L' = \{a^n c^n b^m d^m \mid n, m \geq 0\}$ qui possède la même image de Parikh est algébrique.*

Dans [GS64], Ginsburg et Spanier montrent que si on se restreint au cas des langages bornés, on peut préciser cette condition pour qu'elle soit nécessaire et suffisante.

Théorème 5.3. *Un langage L inclus dans $\sigma_1^* \cdots \sigma_n^*$ est algébrique si et seulement si $\{(k_1, \cdots, k_n) \mid \sigma_1^{k_1} \cdots \sigma_n^{k_n} \in L\}$ est un ensemble semi-linéaire stratifiable.*

Remarquons que les σ_i peuvent être des mots et non des lettres. Ceci provient du fait qu'un tel langage peut être vu comme l'image d'un morphisme envoyant des lettres a_i sur des mots σ_i, et considérer le théorème sur ce langage. Il suffit de remarquer qu'un tel morphisme préserve l'algébricité. Remarquons que l'exemple précédent ne met pas en défaut ce théorème, puisque l'ensemble semi-linéaire exhibé, et qui est celui demandé par le théorème, n'est pas stratifiable. Par contre, l'image de Parikh à considérer pour le langage L' n'est plus la même, mais l'ensemble $\{(n, n, m, m) \mid n, m \geq 0\} = (0, 0, 0, 0) + \mathbb{N}(1, 1, 0, 0) + \mathbb{N}(0, 0, 1, 1)$ qui est stratifiable.

Exemple 5.5. *On considère $\Sigma = \{a, b, c\}$. Le langage $\{a^n b^m c^n \mid n, m \in \mathbb{N}\}$ est algébrique puisque l'ensemble $\{(n_1, n_2, n_3) \in \mathbb{N}^3 \mid n_1 = n_3\}$ est l'ensemble linéaire $P^\star$ où $P = \{(1, 0, 1), (0, 1, 0)\}$ est un ensemble stratifié. Par contre, le langage $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ n'est pas algébrique. Ce qui signifie que l'ensemble semilinéaire $\mathbb{N}(1, 1, 1)$ n'est pas stratifiable.*

Le problème d'algébricité d'un langage borné est donc équivalente au problème de stratifiabilité d'un ensemble semi-linéaire :

Entrée : Un ensemble semi-linéaire S.

Sortie : Si S est stratifiable ou non.

La décidabilité de ce problème est à ce jour toujours ouverte. La difficulté de ce problème provient entre autre du fait qu'un ensemble semi-linéaire n'admet pas qu'une seule représentation en tant qu'ensemble semi-linéaire, comme le montre l'exemple ci-dessous. De plus, il n'y a pas à notre connaissance de représentation canonique des ensembles semi-linéaires qui serait stratifiable si l'ensemble semi-linéaire a une représentation stratifiable.

Exemple 5.6. *On considère l'ensemble linéaire* $L = (0,0,0) + \mathbb{N}(1,1,1) + \mathbb{N}(2,1,0) + \mathbb{N}(0,1,2)$. *Le vecteur* $(1,1,1)$ *a trois composantes non nulles, et l'ensemble* $\{(1,1,1),(2,1,0),(0,1,2)\}$ *n'est donc pas stratifié. Cependant, comme pour tout* n, *on a* $(2n,2n,2n) = n(2,1,0) + n(0,1,2)$ *et* $(2n+1,2n+1,2n+1) = (1,1,1) + n(2,1,0) + n(0,1,2)$, *on a que* L *est l'ensemble semi-linéaire* $((0,0,0)+\mathbb{N}(2,1,0)+\mathbb{N}(0,1,2)) \cup ((1,1,1)+\mathbb{N}(2,1,0)+\mathbb{N}(0,1,2))$, *et l'ensemble* $\{(2,1,0),(0,1,2)\}$ *étant bien stratifié, on a bien que* L *est un ensemble semi-linéaire stratifiable.*

Dans la suite de ce chapitre, on va s'intéresser à un cas particulier de ce problème, à savoir le cas des polyèdres d'entiers, puis on montrera en quoi ce sous-problème est suffisant pour s'intéresser à l'algébricité des traces de systèmes à compteurs plats et de systèmes d'addition de vecteurs.

5.4 Un sous-cas : stratifiabilité des polyèdres d'entiers

Dans ce paragraphe, on montre que le problème de stratifiabilité est décidable pour une sous-classe des ensembles semi-linéaires : les ensembles de solutions entières des systèmes finis d'équations linéaires.

Définition 5.5. *Le* polyèdre d'entiers *associé à une matrice* $\mathbf{A} \in \mathbb{Z}^{n\times d}$ *et à un vecteur* $\vec{b} \in \mathbb{Z}^n$ *est l'ensemble* $\mathcal{P}(\mathbf{A},\vec{b}) = \{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{b}\}$.

Proposition 5.4. *Un polyèdre d'entiers est un ensemble semi-linéaire.*

Cette proposition découle directement du lemme ci-dessous.

Lemme 5.5 ([Sch87, p. 237])**.** *Pour toute matrice* $\mathbf{A} \in \mathbb{Z}^{n\times d}$ *et tout vecteur* $\vec{b} \in \mathbb{Z}^n$, *il existe deux sous-ensembles finis* $\vec{B}$ *et* $\vec{P}$ *de* $\mathbb{N}^d$ *tels que*

$$\{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{b}\} \;=\; \vec{B} + \vec{P}^\star \qquad \text{et} \qquad \vec{P}^\star \;=\; \{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{0}\}$$

Schrijver obtient ce résultat en montrant que l'ensemble des vecteurs satisfaisant $\mathbf{A}\vec{x} \geq 0$ est stable par addition et soustraction de vecteurs dont les composantes sont des sous-déterminant de la matrice $\mathbf{A}$. Ces vecteurs forment l'ensemble des périodes de $\{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{b}\}$ et de $\{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{0}\}$, et pour tout vecteur $\vec{x}$ de $\{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{b}\}$, on peut exhiber un vecteur $\vec{v}$ dont toutes les composantes sont inférieures au plus grand sous-déterminant de $\mathbf{A}$ tel qu'il existe un vecteur $\vec{p}$ formé par l'addition de périodes tel que $\vec{x} = \vec{v} + \vec{p}$. Il y a un nombre fini de vecteurs dont les composantes sont inférieure au plus grand sous-déterminant de $\mathbf{A}$, ce qui permet d'exhiber un ensemble fini de bases.

Exemple 5.7. *On considère la matrice* $\mathbf{A}$ *et le vecteur* $\vec{b}$ *suivants :*

$$\mathbf{A} = \begin{pmatrix} 1 & -1 \\ -1 & 3 \\ 1 & 0 \end{pmatrix} \qquad \vec{b} = (0,2,4)$$

Le polyèdre d'entiers associé $\mathcal{P}(\mathbf{A},\vec{b})$ *est représenté à la figure 5.4. On peut l'exprimer comme l'ensemble semi-linéaire suivant :*

$$\begin{array}{ll} & (4,2) + \mathbb{N}(1,1) + \mathbb{N}(3,1) \\ \cup & (4,3) + \mathbb{N}(1,1) + \mathbb{N}(3,1) \\ \cup & (4,4) + \mathbb{N}(1,1) + \mathbb{N}(3,1) \\ \cup & (6,3) + \mathbb{N}(1,1) + \mathbb{N}(3,1) \end{array}$$

Le *problème de stratifiabilité d'un polyèdre d'entiers* est le suivant :

Entrée : Une matrice $\mathbf{A} \in \mathbb{Z}^{n\times d}$ et un vecteur $\vec{b} \in \mathbb{Z}^n$.

Sortie : Si le polyèdre d'entiers $\mathcal{P}(\mathbf{A},\vec{b})$ est stratifiable ou non.

Dans un premier temps, on va ramener ce problème à un problème de décision impliquant uniquement une matrice $\mathbf{A} \in \mathbb{Z}^{n\times d}$.

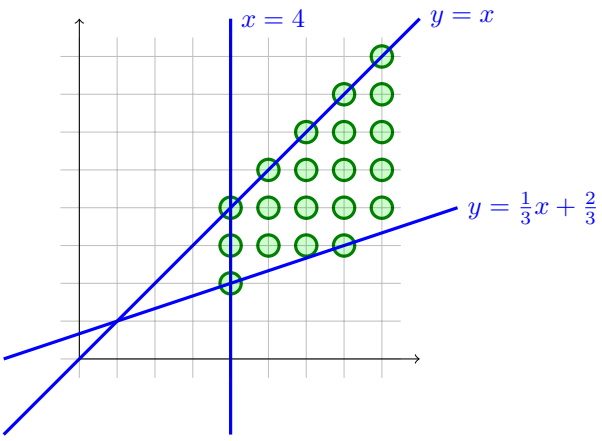


FIGURE 5.4 – Un polyèdre d'entiers

5.4.1 Réduction au cas des cônes

On va réduire le problème de stratifiabilité d'un polyèdre d'entiers à un problème qu'on appellera *problème de bon parenthésage d'un cône*.

On commence par rappeler qu'une relation binaire R sur $\{1, \ldots, d\}$ est dite *bien parenthésée* si pour tout couple (i, j) de R, $i < j$, et qu'il n'existe pas de couple (i, j) et (k, l) dans R tels que $i < k < j < l$.

Définition 5.6. *Le* cône *associé à la matrice* $\mathbf{A} \in \mathbb{Z}^{n \times d}$ *est l'ensemble* $\mathcal{C}(\mathbf{A}) = \{\vec{x} \in \mathbb{Q}^d_{\geq 0} \mid \mathbf{A}\vec{x} \geq \vec{0}\}$.

Étant donné un cône $\mathcal{C}(\mathbf{A}) \subseteq \mathbb{Q}^d_{\geq 0}$ et une relation binaire bien parenthésée R sur $\{1, \ldots, d\}$, on introduit l'ensemble $\mathcal{C}(\mathbf{A})_R$ défini comme suit :

$$\mathcal{C}(\mathbf{A})_R \;=\; \sum_{r \in R} \mathcal{C}(\mathbf{A})_r$$

où, pour chaque couple d'indices $r = (s, t)$ satisfaisant $1 \leq s \leq t \leq d$, l'ensemble $\mathcal{C}(\mathbf{A})_r$ est donné par :

$$\mathcal{C}(\mathbf{A})_r \;=\; \left\{ \vec{x} \in \mathcal{C}(\mathbf{A}) \;\middle|\; \bigwedge_{j \notin \{s,t\}} \vec{x}(j) = 0 \right\}$$

Définition 5.7. *Étant donnée une matrice* $\mathbf{A}$*, on dit que* $\mathcal{C}(\mathbf{A})$ *est bien parenthésé si* $\mathcal{C}(\mathbf{A}) = \bigcup_{R \in \mathcal{R}} \mathcal{C}(\mathbf{A})_R$ *où* $\mathcal{R}$ *désigne l'ensemble de toutes les relations bien parenthésées.*

Le problème *de bon parenthésage d'un cône est le suivant* :

Entrée : Une matrice $\mathbf{A}$.

Sortie : Si $\mathcal{C}(\mathbf{A})$ est bien parenthésé ou non.

Exemple 5.8. *On reprend la matrice* $\mathbf{A}$ *de l'exemple 5.7. Le cône* $\mathcal{C}(\mathbf{A})$ *est représenté à la figure 5.5. Il peut s'exprimer comme* $\mathbb{Q}_{\geq 0}(3, 1) + \mathbb{Q}_{\geq 0}(1, 1)$.

Le théorème suivant permet de ramener le problème de stratifiabilité d'un polyèdre d'entier au problème de bon parenthésage d'un cône.

Théorème 5.6. *Étant donnés une matrice* $\mathbf{A} \in \mathbb{Z}^{n \times d}$ *et un vecteur* $\vec{b} \in \mathbb{Z}^d$*, le polyèdre d'entiers* $\mathcal{P}(\mathbf{A}, \vec{b})$ *est stratifiable si et seulement si il est vide ou que le cône* $\mathcal{C}(\mathbf{A})$ *est bien parenthésé.*

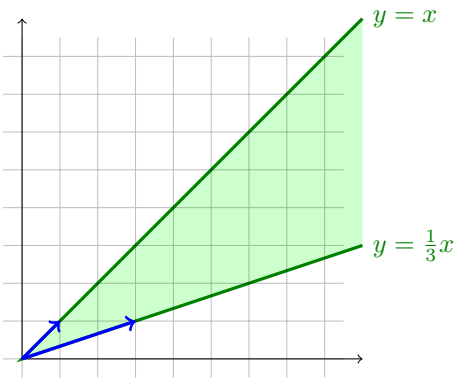


FIGURE 5.5 – Un cône

Pour prouver ce théorème, on montre tout d'abord que la stratifiabilité d'un polyèdre d'entiers peut être réduite au cas particulier d'inégalités linéaires homogènes. Formellement, un *cône d'entiers* est un ensemble de la forme $\{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{0}\}$ où $\mathbf{A} \in \mathbb{Z}^{n \times d}$ est une matrice. Le lemme 5.5 cité précédemment montre que tout cône d'entiers est un ensemble linéaire, et procure une décomposition d'un polyèdre d'entiers en cônes d'entiers.

Étant donné un ensemble fini $\vec{P} = \{\vec{p}_1, \ldots, \vec{p}_k\}$ de vecteurs de $\mathbb{N}^d$, on définit $\vec{P}^\triangleleft$ l'ensemble des combinaisons linéaires de vecteurs de $\vec{P}$ avec des coefficient rationnels non-négatifs, c.a.d, $\vec{P}^\triangleleft = \mathbb{Q}_{\geq 0}\vec{p}_1 + \cdots + \mathbb{Q}_{\geq 0}\vec{p}_k$. Autrement dit, $\vec{P}^\triangleleft$ est défini comme $\vec{P}^\star$ à la différence que $\mathbb{Q}_{\geq 0}\vec{p}_i$ remplace $\mathbb{N}\vec{p}_i$. Observons que $\vec{P}^\triangleleft = \mathbb{Q}_{\geq 0}\vec{P}^\star$ pour tout sous-ensemble fini $\vec{P}$ de $\mathbb{N}^d$. Rappelons la propriété suivante.

Proposition 5.7 (Théorème de Farkas-Minkowski-Weyl)**.** *Un sous-ensemble $\mathcal{C}(\mathbf{A})$ de $\mathbb{Q}^d_{\geq 0}$ est un cône si et seulement si $\mathcal{C}(\mathbf{A}) = \vec{P}^\triangleleft$ où $\vec{P}$ est un sous-ensemble fini de $\mathbb{N}^d$.*

Cette propriété est obtenue d'une part en montrant qu'un cône $\mathcal{C}(\mathbf{A})$ est l'intersection d'un nombre fini d'hyperplans, et d'autre part que si un sous-ensemble est de la forme $\vec{P}^\triangleleft$, on peut construire une matrice $\mathbf{A}$ à partir des générateurs du cône tels que $\vec{P}^\triangleleft = \mathcal{C}(\mathbf{A})$.

On souhaite extraire les directions asymptotiques d'un polyèdre d'entiers. Pour cela, on associe à chaque sous-ensemble $\vec{P} \subseteq \mathbb{N}^d$ un ordre $\sqsubseteq_{\vec{P}}$ sur $\mathbb{N}^d$ défini par $\vec{x} \sqsubseteq_{\vec{P}} \vec{y}$ si $\vec{y} \in \vec{x} + \vec{P}^\star$. Observons que si $\vec{P} = \{\vec{e}_1, \ldots, \vec{e}_d\}$, l'ordre $\sqsubseteq_{\vec{P}}$ coïncide avec l'ordre classique $\leq$ sur $\mathbb{N}^d$, qui est un ordre bien fondé sur $\mathbb{N}^d = \mathbb{N}^d \cap \vec{P}^\triangleleft$, d'après le lemme de Dickson. Cette observation peut être généralisée à tout sous-ensemble fini $\vec{P}$ de $\mathbb{N}^d$, comme suit.

Lemme 5.8 ([HP79])**.** *L'ordre $\sqsubseteq_{\vec{P}}$ sur $\mathbb{N}^d$ est un ordre bien fondé sur $\mathbb{N}^d \cap \vec{P}^\triangleleft$, pour tout sous-ensemble fini $\vec{P}$ de $\mathbb{N}^d$.*

Pour montrer que le problème de la stratifiabilité d'un polyèdre d'entiers est décidable, on décompose les cônes en «parties stratifiables maximales». Intuitivement, ces «parties stratifiables maximales» sont les ensembles $\mathcal{C}(\mathbf{A})_R$. La propriété de stratification de $\mathcal{C}(\mathbf{A})_R$ est exprimée par le lemme suivant.

Lemme 5.9. *Pour tout cône $\mathcal{C}(\mathbf{A}) \subseteq \mathbb{Q}^d_{\geq 0}$ et toute relation binaire bien parenthésée R sur $\{1, \ldots, d\}$, on a que $\mathcal{C}(\mathbf{A})_R = \vec{P}_R^\triangleleft$ pour un sous-ensemble fini stratifié $\vec{P}_R$ de $\mathbb{N}^d$.*

Démonstration. Prenons un cône $\mathcal{C}(\mathbf{A})$ et une relation bien parenthésée R sur $\{1, \ldots, d\}$. Tout d'abord, remarquons que $\mathcal{C}(\mathbf{A})_r$ est un cône pour tout $r = (s, t)$ dans R. En effet, toute contrainte $\vec{x}(j) = 0$ peut être exprimée comme la conjonction de $\vec{x}(j) \geq 0$ et de $-\vec{x}(j) \geq 0$. En ajoutant ces inégalités à la matrice $\mathbf{A}$ pour tout $j \notin \{s, t\}$, on obtient une matrice témoignant que $\mathcal{C}(\mathbf{A})_r$

est un cône. D'après la proposition 5.7 il s'ensuit que $\mathcal{C}(\mathbf{A})_r = \vec{P}_r^{\triangleleft}$ pour un sous-ensemble fini $\vec{P}_r \subseteq \mathbb{N}^d$. Ainsi, $\mathcal{C}(\mathbf{A})_R = \vec{P}_R^{\triangleleft}$ où $\vec{P}_R$ est le sous-ensemble fini de $\mathbb{N}^d$ défini par $\vec{P}_R = \bigcup_{r\in R} \vec{P}_r$. Puisque R est bien parenthésée, on déduit que $\vec{P}_R$ est stratifié, ce qui conclut la preuve du lemme.

□

On peut maintenant donner la preuve du théorème 5.6.

Démonstration. On montre tout d'abord la direction «si» du théorème. Si $\mathcal{P}(\mathbf{A}, \vec{b})$ est vide, il est trivialement stratifiable. Supposons maintenant que $\mathcal{C}(\mathbf{A})$ soit bien parenthésé. D'après le lemme 5.5, le polyèdre d'entiers $\mathcal{P}(\mathbf{A}, \vec{b})$ peut être décomposé en une somme $\mathcal{P}(\mathbf{A}, \vec{b}) = \vec{B} + \vec{H}$ où $\vec{B} \subseteq \mathbb{N}^d$ est un ensemble fini et $\vec{H}$ est le cône d'entiers $\{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{0}\}$. Observons que $\mathcal{C}(\mathbf{A}) = \bigcup_R \mathcal{C}(\mathbf{A})_R$ puisque $\mathcal{C}(\mathbf{A})$ est bien parenthésé, et que $\vec{H} \subseteq \mathcal{C}(\mathbf{A})$ par définition. On en déduit que $\vec{H}$ peut être décomposé en $\vec{H} = \bigcup_R \vec{H}_R$ où $\vec{H}_R = \vec{H} \cap \mathcal{C}(\mathbf{A})_R$. D'après le lemme 5.9, il existe un ensemble fini stratifié $\vec{P}_R \subseteq \mathbb{N}^d$ tel que $\mathcal{C}(\mathbf{A})_R = \vec{P}_R^{\triangleleft}$. Le lemme 5.8 assure alors que $\sqsubseteq_{\vec{P}_R}$ est un bon ordre sur $\mathbb{N}^d \cap \mathcal{C}(\mathbf{A})_R$. Il s'ensuit que l'ensemble des éléments minimaux de $\vec{H}_R$ pour ce bon ordre est un ensemble fini. De plus, en notant $\vec{B}_R$ cet ensemble fini, on obtient que $\vec{H}_R \subseteq \vec{B}_R + \vec{P}_R^{\star}$. On a prouvé l'inclusion $\vec{H} \subseteq \bigcup_R (\vec{B}_R + \vec{P}_R^{\star})$. Puisque l'inclusion inverse est immédiate, l'ensemble $\vec{H}$ est stratifiable. Comme $\mathcal{P}(\mathbf{A}, \vec{b}) = \vec{B} + \vec{H}$, on déduit que $\mathcal{P}(\mathbf{A}, \vec{b})$ est stratifiable.

On prouve maintenant la direction «seulement si» du théorème. Supposons que $\mathcal{P}(\mathbf{A}, \vec{b})$ est non-vide et stratifiable. Puisque $\mathcal{P}(\mathbf{A}, \vec{b})$ est stratifiable, il peut être décomposé en $\bigcup_{i\in I} (\vec{b}_i + \vec{P}_i^{\star})$ où I est fini, $\vec{b}_i \in \mathbb{N}^d$, et $\vec{P}_i \subseteq \mathbb{N}^d$ est un ensemble fini stratifié.

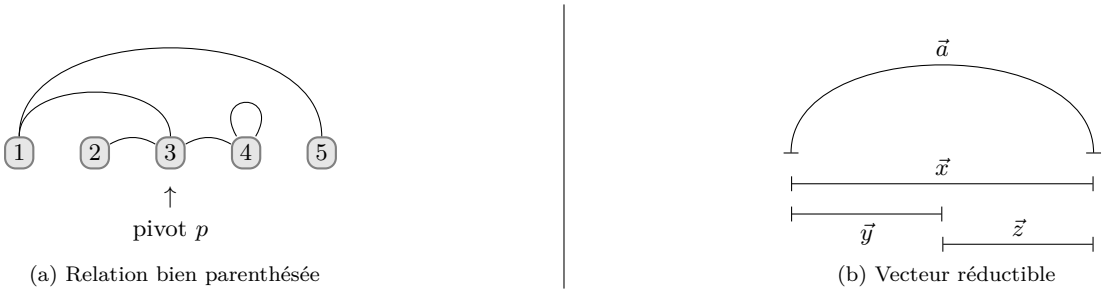
Montrons tout d'abord que $\vec{P}_i^{\star} \subseteq \bigcup_R \mathcal{C}(\mathbf{A})_R$. On définit R_i l'ensemble des couples (s, t), avec $1 \leq s \leq t \leq d$, tels que $\vec{p}(s) \neq 0 \wedge \vec{p}(t) \neq 0$ pour un vecteur $\vec{p} \in \vec{P}_i$. On a déjà vu que R_i est une relation binaire bien parenthésée sur $\{1, \ldots, d\}$, puisque $\vec{P}_i$ est stratifié. Soit $\vec{p} \in \vec{P}_i$. Pour chaque $n \in \mathbb{N}$, on a que $\vec{b}_i + n\vec{p} \in \mathcal{P}(\mathbf{A}, \vec{b})$. Ainsi, $\mathbf{A}\vec{b}_i + n\mathbf{A}\vec{p} \geq \vec{b}$ pour tout $n \in \mathbb{N}$. On en déduit que $\mathbf{A}\vec{p} \geq \vec{0}$, et donc, $\vec{p} \in \mathcal{C}(\mathbf{A})$. Par définition de R_i, on obtient que $\vec{p} \in \mathcal{C}(\mathbf{A})_{R_i}$. Comme $\mathcal{C}(\mathbf{A})_{R_i}$ est clos par addition, on obtient que $\vec{P}_i^{\star} \subseteq \mathcal{C}(\mathbf{A})_{R_i}$.

Pour montrer que $\mathcal{C}(\mathbf{A}) \subseteq \bigcup_R \mathcal{C}(\mathbf{A})_R$, il suffit de montrer que $\mathbb{N}^d \cap \mathcal{C}(\mathbf{A}) \subseteq \bigcup_R \mathcal{C}(\mathbf{A})_R$, puisque les cônes sont clos par multiplication par des rationnels non négatifs. Soit $\vec{x} \in \mathbb{N}^d \cap \mathcal{C}(\mathbf{A})$. Comme $\mathcal{P}(\mathbf{A}, \vec{b})$ est non-vide, il existe $\vec{s} \in \mathcal{P}(\mathbf{A}, \vec{b})$. Pour tout $n \in \mathbb{N}$, on a $\mathbf{A}(\vec{s} + n\vec{x}) \geq \vec{b}$, et donc, $\vec{s} + n\vec{x} \in \mathcal{P}(\mathbf{A}, \vec{b})$. D'après le principe des tiroirs, il existe $i \in I$ et un ensemble infini $N \subseteq \mathbb{N}$ tel que $\vec{s} + n\vec{x} \in \vec{b}_i + \vec{P}_i^{\star}$ pour tout $n \in N$. Le lemme 5.8 entraîne que $\sqsubseteq_{\vec{P}_i}$ est un bon ordre sur $\mathbb{N}^d \cap \vec{P}_i^{\triangleleft}$. Puisque $\vec{x}_n = \vec{s} - \vec{b}_i + n\vec{x}$ est dans $\vec{P}_i^{\star} \subseteq \mathbb{N}^d \cap \vec{P}_i^{\triangleleft}$ pour tout $n \in \mathbb{N}$, on déduit l'existence de $n, m \in N$ tels que $n < m$ et $\vec{x}_n \sqsubseteq_{\vec{P}_i} \vec{x}_m$. Il s'ensuit que $(m-n)\vec{x} = \vec{x}_m - \vec{x}_n \in \vec{P}_i^{\star}$. Puisque $\vec{P}_i^{\star} \subseteq \bigcup_R \mathcal{C}(\mathbf{A})_R$, on obtient que $\vec{x} \in \bigcup_R \mathcal{C}(\mathbf{A})_R$. On vient de montrer que $\mathbb{N}^d \cap \mathcal{C}(\mathbf{A}) \subseteq \bigcup_R \mathcal{C}(\mathbf{A})_R$, et on en conclut que $\mathcal{C}(\mathbf{A})$ est bien parenthésé.

□

Il nous reste à montrer que le problème de bon parenthésage d'un cône est décidable. Pour cela, on peut arguer qu'étant donné un cône $\mathcal{C}(\mathbf{A})$ on peut calculer depuis la matrice $\mathbf{A}$ une formule exprimée dans $\mathsf{FO}(\mathbb{Q}, 0, 1, +, \leq)$ qui exprime l'égalité $\mathcal{C}(\mathbf{A}) = \bigcup_R \mathcal{C}(\mathbf{A})_R$. Puisque que cette théorie est décidable par l'élimination des quantificateurs de Fourier-Motzkin, on en déduit que le problème de bon parenthésage d'un cône est décidable. D'après le théorème 5.6, on en déduit la décidabilité du problème de stratifiabilité d'un polyèdre d'entiers.

Cependant, on souhaite obtenir une mesure plus précise de la complexité de ce problème. Le paragraphe suivante est consacrée à donner un critère de bon parenthésage d'un cône qui est coNP-complet.



(a) Relation bien parenthésée (b) Vecteur réductible

FIGURE 5.6 – Décomposition de relations bien parenthésées et de vecteurs réduits

5.4.2 Un critère de bon parenthésage d'un cône décidable en coNP

Dans ce paragraphe, on donne un critère vérifiant si un cône donné est bien parenthésé ou non. Ce critère mène à une procédure de décision coNP pour la stratifiabilité d'un cône, et similairement pour la stratifiabilité d'un polyèdre d'entiers. On montre aussi que la borne supérieure dans ce dernier cas est atteinte.

On associe à chaque cône $\mathcal{C}(\mathbf{A}) \subseteq \mathbb{Q}^d_{\geq 0}$ l'ensemble $\widehat{\mathcal{C}(\mathbf{A})} = \bigcup_R \mathcal{C}(\mathbf{A})_R$. On rappelle que $\mathcal{C}(\mathbf{A})$ n'est pas bien parenthésé précisément quand $\mathcal{C}(\mathbf{A}) \not\subseteq \widehat{\mathcal{C}(\mathbf{A})}$. On montre que le non bon parenthésage d'un cône $\mathcal{C}(\mathbf{A})$ peut toujours être attesté par un vecteur de $\mathcal{C}(\mathbf{A}) \setminus \widehat{\mathcal{C}(\mathbf{A})}$ d'une forme particulière.

On a besoin des faits suivants. Une illustration du premier lemme est donné à là figure 5.6a.

Lemme 5.10. *Supposons que $d \geq 3$. Une relation binaire R sur $\{1, \ldots, d\}$ est bien parenthésée si et seulement si il et existe un pivot p avec $1 < p < d$ et deux relations binaires bien parenthésées U et V respectivement sur $\{1, \ldots, p\}$ et $\{p, \ldots, d\}$, telles que $R \subseteq \{(1, d)\} \cup U \cup V$.*

Démonstration. On commence par montrer la direction «seulement si» du lemme. Soit R une relation bien parenthésée sur $\{1, \ldots, d\}$. On considère deux cas.

Premièrement, supposons que $(1, j) \in R$ pour un j compris strictement entre 1 et d. Dans ce cas, on choisit pour pivot p le plus grand j satisfaisant la propriété. On définit $U = \{(s, t) \in R \mid t \leq p\}$ et $V = \{(s, t) \in R \mid s \geq p\}$. Étant donné que R est bien parenthésée, U et V sont des relations bien parenthésées sur $\{1, \ldots, p\}$ et $\{p, \ldots, d\}$, respectivement. Montrons que $R \subseteq \{(1, d)\} \cup U \cup V$. Prenons $(s, t) \in R$. Si $t \leq p$ ou $s \geq p$ alors $(s, t) \in U \cup V$.

Il reste le cas où $s < p < t$. Si $s \neq 1$ alors on a que $1 < s < p < t$, ce qui contredit que la supposition que R est bien parenthésée, puisque $(1, p)$ et (s, t) sont tous les deux dans R. Ainsi $s = 1$, ce qui entraîne que $t = d$ par maximalité de p, ainsi, $(s, t) = (1, d)$. Pour le second cas, supposons qu'il n'existe pas de j tel que $(1, j) \in R$ et $1 < j < d$. Dans ce cas, on choisit $p = 2$, $U = \emptyset$ et $V = R \setminus \{(1, d)\}$. Puisque $d \geq 3$, on a $1 < p < d$. Remarquons que $R \subseteq \{(1, d)\} \cup U \cup V$, U est une relation bien parenthésée sur $\{1, \ldots, p\}$ et V est une relation bien parenthésée sur $\{p, \ldots, d\}$.

Dans le cas contraire, considérons un pivot p satisfaisant $1 < p < d$, une relation bien parenthésée U sur $\{1, \ldots, p\}$ et une relation bien parenthésée V sur $\{p, \ldots, d\}$. On montre d'abord que $W = U \cdots V$ est parenthésée. Considérons deux couples (r, t) et (s, u) de W. Si ces deux couples sont tous les deux dans U ou dans V, on obtient que $\neg(r < s < t < u)$ puisque ces deux relations sont bien parenthésées. Si $(r, t) \in U$ et $(s, u) \in V$, on a $t \leq p \leq s$. Si $(r, t) \in V$ et $(s, u) \in U$, on a $u \leq p \leq r$. Ainsi dans tous les cas, on a $\neg(r < s < t < u)$, et donc W est bien parenthésée.

Montrons maintenant que $\{(1, d)\} \cup W$ est bien parenthésée. On considère deux couples (r, t) et (s, u) de cette relation. Une fois de plus, si les deux couples sont tous les deux dans $\{(1, d)\}$ ou dans W, on a $\neg(r < s < t < u)$ puisque ces deux relations sont bien parenthésées. Quand

$(r,t) = (1,d)$ et $(s,u) \in W$, on a $u \leq t$, et quand $(s,u) = (1,d)$ et $(r,t) \in W$, on a $s \leq r$. Ainsi, dans tous les cas, on a $\neg(r < s < t < u)$, et ainsi $\{(1,d)\} \cup W$ est bien parenthésée. Il s'ensuit que toute relation incluse dans $\{(1,d)\} \cup U \cup V$ est bien parenthésée. □

Lemme 5.11. *Pour tout $\vec{x} \in \widehat{\mathcal{C}(\mathbf{A})}$, il existe une relation binaire bien parenthésée R sur*[1] *$\|\vec{x}\|$ telle que $\vec{x} \in \mathcal{C}(\mathbf{A})_R$.*

Démonstration. Pour raccourcir les notations, on note Q la relation $\|\vec{x}\| \times \|\vec{x}\|$. Considérons la relation bien parenthésée R sur $\{1,\ldots,d\}$ qui contient tous les couples (s,s) avec $s \in \|\vec{x}\|$ et tels que $\vec{x} \in \mathcal{C}(\mathbf{A})_R$. De telles relations existent puisque $\vec{x} \in \widehat{\mathcal{C}(\mathbf{A})}$. Parmi celles-ci, on en choisit une telle que $(R \setminus Q)$ est minimale par inclusion. On montre maintenant que R est contenu dans Q, ce qui conclura la preuve du lemme.

Supposons par contradiction que qu'il existe un couple (s,t) dans $R \setminus Q$. Puisque $\vec{x} \in \mathcal{C}(\mathbf{A})_R$, il existe une famille de vecteurs $\{\vec{x}_r\}_{r \in R}$ avec $\vec{x}_r \in \mathcal{C}(\mathbf{A})_r$ telle que $\vec{x} = \sum_{r \in R} \vec{x}_r$. Il s'ensuit que $\vec{x}$ est contenu dans $\mathcal{C}(\mathbf{A})_U + \vec{x}_{(s,t)}$, où U est la relation bien parenthésée sur $\{1,\ldots,d\}$ définie par $U = R \setminus \{(s,t)\}$. Montrons que $\vec{x}_{(s,t)} \in \mathcal{C}(\mathbf{A})_U$. Puisque $(s,t) \notin Q$, on a que $s \notin \|\vec{x}\|$ ou que $t \notin \|\vec{x}\|$. Ce qui signifie que $\vec{x}(s) = 0$ ou que $\vec{x}(t) = 0$. Puisque $\vec{0} \leq \vec{x}_{(s,t)} \leq \vec{x}$, on en déduit que

$$\vec{x}_{(s,t)} = \vec{0} \quad \text{ou} \quad s \in \|\vec{x}\| \ \wedge \ \vec{x}_{(s,t)} \in \mathcal{C}(\mathbf{A})_{(s,s)} \quad \text{ou} \quad t \in \|\vec{x}\| \ \wedge \ \vec{x}_{(s,t)} \in \mathcal{C}(\mathbf{A})_{(t,t)}$$

On rappelle que R contient, par supposition, la relation identité sur $\|\vec{x}\|$. C'est donc également le cas de U, ce qui entraîne que $\vec{x}_{(s,t)} \in \mathcal{C}(\mathbf{A})_U$. Comme $\mathcal{C}(\mathbf{A})_U$ est clos par addition, on obtient que $\vec{x} \in \mathcal{C}(\mathbf{A})_U$, ce qui contredit la minimalité de $(R \setminus Q)$. □

On considère un cône $\mathcal{C}(\mathbf{A}) \subseteq \mathbb{Q}^d_{\geq 0}$. Étant donné un couple d'indices $r = (s,t)$ avec $1 \leq s \leq t \leq d$, on appelle *r-décomposition*[2] tout triplet $(\vec{a},\vec{y},\vec{z})$ de vecteurs de $\mathcal{C}(\mathbf{A})_r \times \mathcal{C}(\mathbf{A}) \times \mathcal{C}(\mathbf{A})$ tels que $\|\vec{y}\| \subseteq \{s,\ldots,p\}$ et $\|\vec{z}\| \subseteq \{p,\ldots,t\}$ pour tout pivot p satisfaisant $s < p < t$. Comme les cônes sont clos par addition, le vecteur $\vec{a}+\vec{y}+\vec{z}$ est dans $\mathcal{C}(\mathbf{A})$ pour toute décomposition $(\vec{a},\vec{y},\vec{z})$. Le lemme suivant montre que l'appartenance à $\widehat{\mathcal{C}(\mathbf{A})}$ est aussi préservée par décomposition.

Lemme 5.12. *Pour toute r-décomposition $(\vec{a},\vec{y},\vec{z})$, on a que $(\vec{a}+\vec{y}+\vec{z}) \in \widehat{\mathcal{C}(\mathbf{A})}$ si $\vec{y} \in \widehat{\mathcal{C}(\mathbf{A})}$ et $\vec{z} \in \widehat{\mathcal{C}(\mathbf{A})}$.*

Démonstration. Supposons que $\vec{y} \in \widehat{\mathcal{C}(\mathbf{A})}$ et $\vec{z} \in \widehat{\mathcal{C}(\mathbf{A})}$. D'après le lemme 5.11, il existe deux relations binaires bien parenthésée U sur $\|\vec{y}\|$ et V sur $\|\vec{z}\|$ telles que $\vec{y} \in \mathcal{C}(\mathbf{A})_U$ et $\vec{z} \in \mathcal{C}(\mathbf{A})_V$. Puisque $(\vec{a},\vec{y},\vec{z})$ est une r-décomposition, il existe un pivot p avec $s < p < t$, où $r = (s,t)$, tel que $\|\vec{y}\| \subseteq \{s,\ldots,p\}$ et $\|\vec{z}\| \subseteq \{p,\ldots,t\}$. Il s'ensuit que le champs[3] de U et celui de V sont respectivement contenus dans $\{s,\ldots,p\}$ et $\{p,\ldots,t\}$. On déduit du lemme 5.10 que la relation binaire $R = \{(s,t)\} \cup U \cup V$ est bien parenthésée. Observons que $\vec{a}$, $\vec{y}$ et $\vec{z}$ sont tous dans $\mathcal{C}(\mathbf{A})_R$. Puisque $\mathcal{C}(\mathbf{A})_R$ est clos par addition, on déduit que $(\vec{a}+\vec{y}+\vec{z}) \in \mathcal{C}(\mathbf{A})_R$, ce qui conclut la preuve du lemme.

□

On dit qu'un vecteur $\vec{x} \in \mathcal{C}(\mathbf{A})$ est *réductible*[4] quand $\|\vec{x}\|$ a au plus deux éléments ou qu'il existe un couple d'indices (s,t) de $\|\vec{x}\|$, avec $1 \leq s \leq t \leq d$, et une (s,t)-décomposition $(\vec{a},\vec{y},\vec{z})$ tels que $\vec{x} = \vec{a} + \vec{y} + \vec{z}$. La dernière condition est représentée à la figure 5.6b. Notons que cette condition implique que $\|\vec{x}\| \subseteq \{s,\ldots,t\}$. Un vecteur $\vec{x} \in \mathcal{C}(\mathbf{A})$ est dit *irréductible*[4] quand il n'est pas réductible. Le théorème suivant caractérise quels cônes sont bien parenthésés, en termes de vecteurs irréductibles. Avant cela, illustrons ces notions avec quelques exemples.

1. Rappelons que $\|\vec{v}\|$ désigne le support de $\vec{v}$, c.à.d, l'ensemble des indices i tels que $\vec{v}(i) \neq 0$.
2. Cette notion est définie relativement à un cône $\mathcal{C}(\mathbf{A})$, laissé ici implicite pour ne pas surcharger les notations.
3. Le *champs* d'une relation binaire est l'union de son domaine et de son image.

Exemple 5.9. *Soient $\vec{X}$ et $\vec{Y}$ les cônes donnés par $\vec{X} = \{\vec{x} \in \mathbb{Q}^3_{\geq 0} \mid \vec{x}(1) = \vec{x}(3)\}$ et $\vec{Y} = \{\vec{x} \in \mathbb{Q}^3_{\geq 0} \mid \vec{x}(1) = \vec{x}(2) = \vec{x}(3)\}$. Le vecteur $(1,0,1)$ est réductible pour les deux cônes, puisqu'il contient seulement deux composantes non-nulles. Le vecteur $(1,1,1)$ est réductible pour $\vec{X}$. Cela est attesté par la $(1,3)$-décomposition $(\vec{a},\vec{y},\vec{y})$ où $\vec{a} = (1,0,1)$ et $\vec{y} = (0,0.5,0)$. Le même vecteur $(1,1,1)$ est irréductible pour $\vec{Y}$.*

Théorème 5.13. *Un cône est bien parenthésé si et seulement si il ne contient pas de vecteur irréductible.*

Démonstration. On considère un cône $\mathcal{C}(\mathbf{A}) \subseteq \mathbb{Q}^d_{\geq 0}$. On montre d'abord la direction «seulement si» du théorème. Supposons que $\mathcal{C}(\mathbf{A})$ est bien parenthésé, et prenons $\vec{x} \in \mathcal{C}(\mathbf{A})$. Si $\|\vec{x}\|$ contient au plus deux éléments, alors $\vec{x}$ est trivialement réductible. Supposons au contraire que $\vec{x}$ a au moins trois composantes non-nulles. Puisque $\mathcal{C}(\mathbf{A}) \subseteq \widehat{\mathcal{C}(\mathbf{A})}$, le lemme 5.11 nous donne qu'il existe une relation binaire bien parenthésée R sur $\|\vec{x}\|$ telle que $\vec{x} \in \mathcal{C}(\mathbf{A})_R$. Par ailleurs R n'est pas vide puisque $\vec{x} \neq \vec{0}$. On note F le champs [5] de R, et on définit $s = \min F$ et $t = \max F$. Observons que s et t sont tous les deux dans $\|\vec{x}\|$. Notons également que $t \geq s+2$ puisque $\vec{x}$ contient au moins trois composantes non-nulles. On déduit du lemme 5.10 qu'il existe un pivot p avec $s < p < t$, une relation binaire bien parenthésée U sur $\{s,\ldots,p\}$ et une autre V sur $\{p,\ldots,t\}$ telles que $R \subseteq \{(s,t)\} \cup U \cup V$. On en déduit que $\vec{x} \in (\mathcal{C}(\mathbf{A})_{(s,t)} + \mathcal{C}(\mathbf{A})_U + \mathcal{C}(\mathbf{A})_V)$, ce qui entraîne que $\vec{x}$ est réductible.

On montre maintenant la direction «si» du théorème. Supposons que $\mathcal{C}(\mathbf{A})$ n'est pas bien parenthésé. Cela signifie que $\mathcal{C}(\mathbf{A}) \not\subseteq \widehat{\mathcal{C}(\mathbf{A})}$. Parmi les vecteurs $\vec{x}$ de $\mathcal{C}(\mathbf{A}) \setminus \widehat{\mathcal{C}(\mathbf{A})}$, on en choisit un tel que $\|\vec{x}\|$ soit minimal pour l'inclusion. Montrons que ce vecteur est irréductible. Par contradiction, supposons qu'il est réductible. $\|\vec{x}\|$ contient au moins trois éléments puisque $\vec{x} \in (\mathcal{C}(\mathbf{A}) \setminus \widehat{\mathcal{C}(\mathbf{A})})$. Il s'ensuit qu'il existe un couple (s,t) d'indices de $\|\vec{x}\|$, avec $1 \leq s \leq t \leq d$, et une (s,t)-décomposition $(\vec{a},\vec{y},\vec{z})$ tels que $\vec{x} = \vec{a} + \vec{y} + \vec{z}$. On a directement que $\|\vec{y}\|$ et $\|\vec{z}\|$ sont strictement contenus dans $\|\vec{x}\|$. Par minimalité de $\vec{x}$, on déduit que $\vec{y}$ et $\vec{z}$ sont dans $\widehat{\mathcal{C}(\mathbf{A})}$. On en déduit par le lemme 5.12 que $\vec{x} \in \widehat{\mathcal{C}(\mathbf{A})}$, ce qui contredit la supposition que $\vec{x}$ est dans $\mathcal{C}(\mathbf{A}) \setminus \widehat{\mathcal{C}(\mathbf{A})}$.

□

Ce théorème nous permet de réduire le problème de bon parenthésage d'un cône à la question de savoir si un cône contient uniquement des vecteurs réductibles. Dans le reste de ce paragraphe, on explique comment résoudre ce dernier problème dans coNP. Considérons une matrice $\mathbf{A} \in \mathbb{Z}^{n\times d}$ encodée en binaire, et prenons $\mathcal{C}(\mathbf{A}) = \{\vec{x} \in \mathbb{Q}^d_{\geq 0} \mid \mathbf{A}\vec{x} \geq \vec{0}\}$. On construit, en temps polynomial en la taille de $\mathbf{A}$, une formule sans quantificateurs [4] $\rho(\vec{x})$ exprimée dans $\mathsf{FO}(\mathbb{Q},0,1,+,\leq)$ qui est valide si et seulement si $\mathcal{C}(\mathbf{A})$ ne contient que des vecteurs réductibles. On en déduira une borne supérieur coNP pour le problème de bon parenthésage d'un cône, puisque la satisfiabilité d'une formule sans quantificateurs exprimée dans $\mathsf{FO}(\mathbb{Q},0,1,+,\leq)$ est résoluble en NP (voir, par exemple, [Sch87, p. 120]). Premièrement, on construit un formule $\varphi(\vec{x})$ avec quantificateurs, dont les modèles sont précisément tous les vecteurs réductibles de $\mathcal{C}(\mathbf{A})$. Soit $\mathbf{B}$ la matrice de $\mathbb{Z}^{(n+d)\times d}$ obtenue à partir de $\mathbf{A}$ en collant la matrice identité en dessous de $\mathbf{A}$. On note que $\mathcal{C}(\mathbf{A}) = \{\vec{x} \in \mathbb{Q}^d \mid \mathbf{B}\vec{x} \geq \vec{0}\}$. La formule $\varphi(\vec{x})$ est :

$$\mathbf{B}\vec{x} \geq \vec{0} \ \wedge \bigvee_{1\leq s\leq t\leq d} \left[\left(\bigwedge_{i\notin\{s,t\}} \vec{x}(i) = 0 \right) \vee (\vec{x}(s) > 0 \wedge \vec{x}(t) > 0 \wedge \psi_{\mathbf{B},s,t}(\vec{x})) \right]$$

où pour tout couple d'indices (s,t) avec $1 \leq s \leq t \leq d$, la formule $\psi_{\mathbf{B},s,t}(\vec{x})$, donnée ci-dessous, exprime qu'il existe une (s,t)-décomposition $(\vec{a},\vec{y},\vec{z})$ telle que $\vec{x} = \vec{a}+\vec{y}+\vec{z}$. La formule $\psi_{\mathbf{B},s,t}(\vec{x})$

4. on suppose que toutes les constantes entières des formules sont encodées en binaire.

est :

$$\left(\bigwedge_{i<s \vee i>t} \vec{x}(i) = 0\right) \wedge \bigvee_{p=s+1}^{t-1} \exists\mu\,\exists\nu\,\exists\pi \begin{bmatrix} \mathbf{B}(\mu\vec{e}_s + \nu\vec{e}_t) & \geq & \vec{0} \\ \mathbf{B}\left(\sum_{s\leq i<p} \vec{x}(i)\vec{e}_i - \mu\vec{e}_s + \pi\vec{e}_p\right) & \geq & \vec{0} \\ \mathbf{B}\left(\sum_{p\leq i\leq t} \vec{x}(i)\vec{e}_i - \pi\vec{e}_p - \nu\vec{e}_t\right) & \geq & \vec{0} \end{bmatrix}$$

Ici, il est sous-entendu que la sous-formule entre crochets remplace la conjonction des trois systèmes d'inégalités linéaires. On peut vérifier facilement que pour tout vecteur $\vec{x} \in \mathbb{Q}^d$, la formule $\varphi(\vec{x})$ est vraie si et seulement si $\vec{x}$ est un vecteur réductible de $\mathcal{C}(\mathbf{A})$. Remarquons que chaque clause $\exists\mu\exists\nu\exists\pi\,[\cdots]$ contient un nombre constant de quantificateurs, en l'occurence trois. On peut donc, par l'élimination des quantificateurs de Fourier-Motzkin, transformer, en temps polynomial, chaque formule $\psi_{\mathbf{B},s,t}(\vec{x})$ en une formule sans quantificateurs équivalente $\psi'_{\mathbf{B},s,t}(\vec{x})$. On note $\varphi'(\vec{x})$ la formule obtenue à partir de la définition de $\varphi(\vec{x})$ en replaçant chaque $\psi_{\mathbf{B},s,t}(\vec{x})$ par $\psi'_{\mathbf{B},s,t}(\vec{x})$. La formule désirée $\rho(\vec{x})$ est $(\mathbf{B}\vec{x} \geq \vec{0}) \Rightarrow \varphi'(\vec{x})$. On a ainsi montré le théorème suivant :

Théorème 5.14. *Le problème de bon parenthésage d'un cône peut être résolu en coNP.*

On a maintenant tous les ingrédients nécessaires pour montrer le corollaire suivant :

Corollaire 5.15. *Le problème de stratification d'un polyèdre d'entiers est coNP-complet.*

Démonstration. On commence par rappeler que le problème du vide pour les polyèdres d'entiers est coNP-complet (voir, par exemple, [Sch87, p. 245]). Ce problème demande si, étant donné une matrice $\mathbf{A} \in \mathbb{Z}^{n\times d}$ et un vecteur $\vec{b} \in \mathbb{Z}^d$, tous les deux encodés en binaire, si le polyèdre d'entiers $\{\vec{x} \in \mathbb{N}^d \mid \mathbf{A}\vec{x} \geq \vec{b}\}$ est vide.

On montre maintenant le corollaire. La borne supérieur se déduit des théorèmes 5.6 et 5.14, et de la clôture par union de coNP. La borne inférieure s'obtient par réduction du problème du vide pour un polyèdre d'entiers. Premièrement, observons qu'en augmentant d de 3 et en modifiant légèrement $(\mathbf{A}, \vec{b})$, le problème du vide pour un polyèdre d'entiers peut être réduit, en temps linéaire, à un cas particulier d'un polyèdre d'entiers $\mathcal{P}(\mathbf{A}, \vec{b})$ satisfaisant la condition suivante :

$$\mathcal{P}(\mathbf{A}, \vec{b}) = \emptyset \qquad \text{ou} \qquad \{(\vec{x}(1), \vec{x}(2), \vec{x}(3)) \mid \vec{x} \in \mathcal{P}(\mathbf{A}, \vec{b})\} \;=\; \mathbb{N}(1,1,1) \tag{5.4.1}$$

On rappelle que l'ensemble linéaire $\mathbb{N}(1,1,1)$ n'est pas stratifiable (c.f. exemple 5.5). On déduit par le lemme 5.1 que tout polyèdre d'entiers satisfaisant (5.4.1) est vide si et seulement si il est stratifiable. On a ainsi réduit, en temps linéaire, le problème du vide pour un polyèdre d'entiers au problème de stratification pour un polyèdre d'entiers.

□

5.5 Algébricité de traces de systèmes à compteurs

Dans ce paragraphe, on s'intéresse au langages des traces des systèmes à compteurs. On va chercher à résoudre le problème d'algébricité pour un système à compteurs. Le problème est le suivant :

Entrée : Un système à compteurs $\mathcal{S} = \langle \Theta, \vec{c}_{\text{Init}} \rangle$.

Sortie : Si $\mathcal{L}(\mathcal{S})$ est algébrique ou non.

	Plats	Cas général
Système d'addition de vecteurs	coNP-complet	EXPSpace-complet
Systèmes à compteurs	coNP-complet	indécidable

FIGURE 5.7 – Complexité du problème d'algébricité.

Dans le cas général, ce problème est indécidable : en effet, toute machine de Turing est équivalente à un système à compteurs, et le problème d'algébricité pour une machine de Turing est indécidable.

On va donc se ramener à des cas particuliers, à savoir le problème d'algébricité des systèmes à compteurs plats et les systèmes d'addition de vecteurs pour lesquels ce problème est décidable. Au final, on obtient les résultats qui sont présentés en figure 5.7.

5.5.1 Algébricité des traces de systèmes à compteurs plats

Nous allons maintenant montrer en quoi le problème de stratifiabilité d'un polyèdre d'entiers peut être utilisé pour résoudre le problème de l'algébricité du langage des traces d'un système à compteurs plat. Pour cela, on va simplement montrer que l'image de Parikh d'un tel langage est toujours un polyèdre d'entiers et conclure en utilisant le résultat du paragraphe précédent.

On considère d'abord le problème suivant, appelé problème de l'algébricité d'un système à compteurs plat linéaire :

Entrée : Un système à compteurs $\mathcal{S} = \langle \Theta, \vec{c}_{\text{Init}} \rangle$ et une suite finie $\tau = w_1, \sigma_1, \cdots, w_d, \sigma_d$ de mots de Θ^*.

Sortie : Si le langage $\mathcal{L}(\mathcal{S}) \cap w_1\sigma_1^+ \cdots w_d\sigma_d^+$ est algébrique.

Le théorème 5.3 nous dit que ce langage est algébrique si et seulement si l'ensemble $N_{\mathcal{S},\tau} = \{(n_1, \cdots, n_d \mid w_1\sigma_1^{n_1} \cdots w_d\sigma_d^{n_d}\}$ est un ensemble semilinéaire stratifiable. Il nous suffit donc de montrer le théorème suivant.

Proposition 5.16. *Pour tout système à compteurs $\mathcal{S}$ et suite de mots τ, l'ensemble $N_{\mathcal{S},\tau}$ est un polyèdre d'entiers.*

Pour prouver cette proposition, on utilise d'abord le lemme suivant :

Lemme 5.17. *Il existe un algorithme en temps polynomial qui, étant donné un système à compteur $\mathcal{S}$ et un mot $\sigma \in \Theta^*$, calcule une matrice $\mathbf{A} \in \mathbb{Z}^{n \times c}$ et trois vecteurs $\vec{a}, \vec{b} \in \mathbb{Z}^n$ et $\vec{c} \in \mathbb{Z}^n$ tels que :*

$$\vec{x} \xrightarrow{\sigma^n} \vec{y} \iff \mathbf{A}\vec{x} + n\vec{a} \geq \vec{b} \wedge \vec{y} = \vec{x} + n\vec{v}$$

Pour montrer le lemme 5.17, on donne d'abord une méthode pour encoder la relation binaire $\xrightarrow{\sigma}$ avec $\sigma \in \Theta^*$ comme une relation binaire $\xrightarrow{\theta}$ avec $\theta \in \Theta$.

Lemme 5.18. *Il existe un algorithme en temps polynomial calculant une transition $\theta \in \Theta$ pour un nombre de compteurs $c \in \mathbb{N}$ et un mot $\sigma \in \Theta^*$ tels que $\xrightarrow{\sigma}$ est égal à $\xrightarrow{\theta}$.*

Démonstration. Supposons que $\sigma = \theta_1 \dots \theta_k$ avec $\theta_j = (\mathbf{A}_j, \vec{b}_j, \vec{v}_j)$ où $\mathbf{A}_j \in \mathbb{Z}^{m_j \times c}$, $\vec{b}_j \in \mathbb{Z}^{m_j}$, et $\vec{v}_j \in \mathbb{Z}^c$. Soit $\vec{v} = \vec{v}_1 + \cdots + \vec{v}_k$. Observons que $\vec{x} \xrightarrow{\sigma} \vec{y}$ si et seulement si, $\vec{y} = \vec{x} + \vec{v}$ et le système linéaire suivant est vrai, avec $\vec{x}_j = \vec{x} + \vec{v}_1 + \cdots + \vec{v}_{j-1}$

$$\bigwedge_{j=1}^{k} \mathbf{A}_j\vec{x}_j \geq \vec{b}_j \ \wedge \ \vec{x}_j \geq \vec{0}$$

Un tel système linéaire est noté $\mathbf{A}\vec{x} \geq \vec{b}$. La transition $\theta = (\mathbf{A}, \vec{b}, \vec{v})$ vérifie le lemme.

□

Lemme 5.19 ([BFLP08])**.** *Soit $\theta = (\mathbf{A}, \vec{b}, \vec{v})$ une transition de Θ, $n \in \mathbb{N}_1$, et $\vec{x}, \vec{y} \in \mathbb{N}^c$. On a $\vec{x} \xrightarrow{\theta^n} \vec{y}$ si et seulement si la condition suivante est vraie :*

$$\mathbf{A}\vec{x} \geq \vec{b} \wedge \mathbf{A}\vec{x} + (n-1)\mathbf{A}\vec{v} \geq \vec{b} \wedge \vec{y} = \vec{x} + n\vec{v}$$

Démonstration. Soit $\vec{x}, \vec{y} \in \mathbb{N}^c$ et $n \geq 1$.

Supposons d'abord que $\vec{x} \xrightarrow{\theta^n} \vec{y}$. Dans ce cas $\vec{y} = \vec{x} + n\vec{v}$. Étant donné que $n \geq 1$, on déduit que $\vec{x} \xrightarrow{\theta} \vec{x} + \vec{v}$ et $\vec{y} - \vec{v} \xrightarrow{\theta} \vec{y}$. La première relation montre que $\mathbf{A}\vec{x} \geq \vec{b}$, et la seconde, avec le fait que $\vec{y} - \vec{v} = \vec{x} + (n-1)\vec{v}$, montre que $\mathbf{A}\vec{x} + (n-1)\mathbf{A}\vec{v} \geq \vec{b}$.

Réciproquement, supposons que $\mathbf{A}\vec{x} \geq \vec{b}$, $\mathbf{A}\vec{x} + (n-1)\mathbf{A}\vec{v} \geq \vec{b}$, et $\vec{y} = \vec{x} + n\vec{v}$. Prouvons que $\vec{x} \xrightarrow{\theta^n} \vec{y}$. On introduit la suite $\vec{x}_0, \ldots, \vec{x}_n$ de vecteurs de $\mathbb{Z}^c$ définie par $\vec{x}_j = \vec{x} + j\vec{v}$. Remarquons que pour tout $j \in \{0, \ldots, n\}$ on a :

$$\vec{x}_j = \vec{x} + j\vec{v} = \frac{n-j}{n}\vec{x} + \frac{j}{n}\vec{y}$$

Puisque $\vec{x}, \vec{y} \geq \vec{0}$, on a $\vec{x}_j \geq \vec{0}$. Remarquons maintenant que pour tout $j \in \{0, \ldots, n-1\}$, on a :

$$\begin{aligned}\mathbf{A}\vec{x}_j &= \mathbf{A}\vec{x} + j\mathbf{A}\vec{v} \\ &= \frac{n-1-j}{n-1}\mathbf{A}\vec{x} + \frac{j}{n-1}(\mathbf{A}\vec{x} + (n-1)\mathbf{A}\vec{v})\end{aligned}$$

Puisque $\mathbf{A}\vec{x} \geq \vec{b}$ et $\mathbf{A}\vec{x} + (n-1)\mathbf{A}\vec{v} \geq \vec{b}$, on déduit que $\mathbf{A}\vec{x}_j \geq \vec{b}$. Ainsi $\vec{x}_j \xrightarrow{\theta} \vec{x}_{j+1}$ pour tout $j \in \{0, \ldots, n-1\}$. On a montré que $\vec{x}_0 \xrightarrow{\theta^n} \vec{x}_n$. Comme $\vec{x}_0 = \vec{x}$ et $\vec{x}_n = \vec{y}$, la preuve est terminée. □

La preuve du lemme 5.17 découle directement des deux lemmes précédents.

La propriété 5.16 est ensuite prouvé par le corollaire suivant :

Corollaire 5.20. *Il existe un algorithme en temps polynomial qui, étant donné un système à compteur $\mathcal{S}$ et une suite de mots $\tau = w_1, \sigma_1, \cdots, w_d, \sigma_d$ de Θ^* calcule une matrice $\mathbf{A} \in \mathbb{Z}^{n \times d}$ et un vecteur $\vec{b} \in \mathbb{Z}^n$ tels que :*

$$w_1\sigma_1^{n_1} \cdots w_d\sigma_d^{n_d} \in \mathcal{L}(\mathcal{S}) \iff \mathbf{A}(n_1, \cdots, n_d) \geq \vec{b}$$

pour tous $n_1, \cdots, n_d \in \mathbb{N}_1$.

Démonstration. Le lemme 5.17 nous donne en temps polynomial un tuple $(\mathbf{A}_i, \vec{a}_i, \vec{b}_i, \vec{v}_i)$ tel que pour chaque $n \in \mathbb{N}_1$ et tous $\vec{x}, \vec{y} \in \mathbb{N}^c$ on a $\vec{x}_i \xrightarrow{\sigma_i^n} \vec{y}$ si et seulement si $\mathbf{A}_i\vec{x} + n\vec{a}_i \geq \vec{b}_i$ et $\vec{y} = \vec{x} + n_i\vec{v}_i$. En temps polynomial, on calcule des transitions $\theta_i = (\mathbf{B}_i, \vec{c}_i, \vec{u}_i)$ telle que la relation binaire $\xrightarrow{w_i}$ est égale à $\xrightarrow{\theta_i}$. Le mot $w_1\sigma_1^{n_1} \cdots w_d\sigma_d^{n_d}$ est une trace depuis $\vec{c}_{\text{Init}}$ avec $n_1, \cdots, n_d \in \mathbb{N}_1$ si et seulement si le système d'équations linéaires suivant est satisfiable :

$$\bigwedge_{i=1}^{d} \mathbf{B}_i\vec{y}_i \geq \vec{c}_i \wedge \mathbf{A}_i\vec{x}_i + n_i\vec{a}_i \geq \vec{b}_i$$

où $\vec{y}_i = \vec{c}_{\text{Init}} + \Sigma_{1 \leq j < i}(\vec{u}_j + n_j\vec{v}_j)$ et $\vec{x}_i = \vec{y}_i + \vec{u}_i$.

Il suffit maintenant de remarquer qu'un tel système linéaire peut être écrit comme un système linéaire de la forme $\mathbf{A}(n_1, \cdots, n_d) \geq \vec{b}$.

□

On en déduit le théorème suivant :

Théorème 5.21. *Le problème de l'algébricité d'un système à compteurs plat linéaire est coNP-complet.*

Démonstration. Puisque le langage est algébrique si et seulement si le polyèdre entier $N_{\mathcal{S},\tau}$ est stratifiable, on déduit du corollaire 5.15 que le problème de l'algébricité d'un système à compteurs plat est dans coNP.

On montre que le problème est coNP-difficile par une réduction directe depuis le problème de stratifiabilité pour les polyèdres d'entiers, qui est coNP-difficile, d'après le corollaire 5.15.

Lemme 5.22. *Le problème d'algébricité pour les systèmes à compteurs plats linéaires est coNP-difficile.*

Démonstration. On considère un polyèdre d'entiers $\mathcal{P}(\mathbf{A},\vec{b})$. On introduit une suite de mots $w_1,\sigma_1,\ldots,w_d,\sigma_d$ de Θ^* où $d=c+1$, et $w_1,\ldots,w_c$ et σ_d sont égaux au mot vide. Le mot w_d est définit comme le triplet $(\mathbf{A},\vec{b}+\mathbf{A}(1,\ldots,1),\vec{0})$. Les autres mots, c.a.d, les mots σ_i avec $1\leq i\leq c$, sont définit comme des triplets $(\mathbf{0},\vec{0},\vec{e_i})$. Observons que $\vec{x}\xrightarrow{\sigma_i^n}\vec{y}$ si et seulement si $\vec{y}=\vec{x}+n\vec{e_i}$. On introduit le langage $L=\mathcal{L}(\mathcal{S})\cap w_1\sigma_1^+\cdots w_d\sigma_d^+$ où $\vec{c}_{\text{Init}}=\vec{0}$. Grâce au lemme précédent, et aux égalités suivantes, on déduit que L est algébrique si et seulement si $(1,\ldots,1)+(\mathcal{P}(\mathbf{A},\vec{b})\times\mathbb{N})$ est stratifiable. Le lemme 5.1 montre que cet ensemble est stratifiable si et seulement si $\mathcal{P}(\mathbf{A},\vec{b})$ est stratifiable. En particulier, on a réduit la stratifiabilité de $\mathcal{P}(\mathbf{A},\vec{b})$ au problème de l'algébricité d'un système à compteurs plat linéaire. Ce dernier problème est donc coNP-difficile.

$$\begin{aligned}&\{(n_1,\ldots,n_d)\in\mathbb{N}_1^d \mid w_1\sigma_1^{n_1}\cdots w_d\sigma_d^{n_d}\in L\}\\&=\{(n_1,\ldots,n_d)\in\mathbb{N}_1^d \mid \mathbf{A}((-1+n_1,\ldots,-1+n_c))\geq\vec{b}\}\\&=(1,\ldots,1)+(\mathcal{P}(\mathbf{A},\vec{b})\times\mathbb{N})\end{aligned}$$

□

Exemple 5.10. *On considère le système à compteurs* $\mathcal{S}=\langle\Theta,\vec{0}\rangle$*, avec* $\Theta=\{\vec{a}_1,\vec{a}_2,\vec{a}_3\}$ *et*

- $\vec{a}_1=\begin{pmatrix}0 & 0\end{pmatrix},(0),(1,1)$
- $\vec{a}_2=\begin{pmatrix}1 & 0\end{pmatrix},(4),(-1,1)$
- $\vec{a}_3=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix},(3,10),(1,-1)$

On considère la suite de mots $\tau=\varepsilon,\vec{a}_1,\varepsilon,\vec{a}_2,\varepsilon,\vec{a}_3$*.*

La procédure qu'on vient de décrire nous donne que $\mathcal{L}(\mathcal{S})\cap\vec{a}_1^*\vec{a}_2^*\vec{a}_3^*$ *est le polyèdre d'entier* $\mathcal{P}(\mathbf{A},\vec{b})$ *avec*

$$\mathbf{A}=\begin{pmatrix}1 & 0 & 0\\ 1 & 0 & 0\\ 1 & -1 & 0\\ 1 & 1 & 0\\ 1 & -1 & 1\\ 1 & 1 & -1\end{pmatrix}\qquad\vec{b}=(0,0,3,0,2,9)$$

□

On considère maintenant le problème de l'algébricité d'un système à compteurs plat proprement dit :

Entrée : Un système à compteurs $\mathcal{S}=\langle\Theta,\vec{c}_{\text{Init}}\rangle$ et un automate $\mathcal{A}$ reconnaissant un langage borné régulier.

Sortie : Si le langage $\mathcal{L}(\mathcal{S})\cap L(\mathcal{A})$ est algébrique.

Théorème 5.23. *Le problème de l'algébricité d'un système à compteurs plat est coNP-complet, et le reste si on se restreint au cas des systèmes d'addition de vecteurs plats.*

Pour montrer ce théorème, on considère le lemme suivant :

Lemme 5.24. *Soit $\mathcal{A}$ un automate fini. Si $L(\mathcal{A})$ est borné, il est l'union des langages $w_1\sigma_1^+ \cdots w_d\sigma_d^+$ tels que $\mathcal{A}$ contienne un calcul acceptant $q_0 \xrightarrow{w_1} q_1 \xrightarrow{\sigma_1} q_1 \cdots q_{d-1} \xrightarrow{w_d} q_d \xrightarrow{\sigma_d} q_d$ avec $d + |w_1\sigma_1 \cdots w_d\sigma_d| \leq 6|Q|^3$.*

Démonstration. Supposons que $L(\mathcal{A})$ est borné, et soit $u \in L(\mathcal{A})$. On décompose les calculs acceptant u dans le but d'en trouver un qui satisfait la condition du lemme. Formellement, une *décomposition linéaire plate* de u est une suite finie $(q_0, w_1, q_1, \sigma_1, \ldots, w_d, q_d, \sigma_d)$ telle que $u \in w_1\sigma_1^+ \cdots w_d\sigma_d^+$, $\mathcal{A}$ contienne un calcul acceptant $q_0 \xrightarrow{w_1} q_1 \xrightarrow{\sigma_1} q_1 \cdots q_{d-1} \xrightarrow{w_d} q_d \xrightarrow{\sigma_d} q_d$, et $|\sigma_i| \leq |Q|$ pour tout $i \in \{1, \ldots, d\}$. Le *rang* de cette décomposition est $(|w_1 \cdots w_d|, d)$. Les rangs sont ordonnés par l'ordre lexicographique sur $\mathbb{N}^2$. Remarquons que u admet une décomposition linéaire plate. En effet, puisque $u \in L(\mathcal{A})$, il existe un calcul acceptant $q \xrightarrow{u} r$ dans $\mathcal{A}$, et la suite (q, u, r, ε) est une décomposition de u.

Considérons une décomposition linéaire plate $(q_0, w_1, q_1, \sigma_1, \ldots, w_d, q_d, \sigma_d)$ de u de rang minimal. Par définition, il existe d entiers positifs $n_1, \ldots, n_d$ tels que $u = w_1\sigma_1^{n_1} \cdots w_d\sigma_d^{n_d}$. Il est évident que $\sigma_i^{n_i}$ est non-vide pour tout $i \in \{1, \ldots, d-1\}$. Dans le cas contraire, on peut obtenir une décomposition de rang plus petit en retirant σ_i et en réunissant w_i et w_{i+1}. De même, chaque w_i a une longueur d'au plus $|Q|$. Dans le cas contraire, on peut obtenir une décomposition de rang plus petit en retirant un circuit simple de w_i. Pour conclure la preuve du lemme, il nous reste à montrer que $d \leq |Q|^2 + 1$.

Supposons, par contradiction, que $d > |Q|^2 + 1$. Le principe des tiroirs montre qu'il existe un état p est répété au moins $|Q| + 1$ fois parmi $\{q_1, \ldots, q_{d-1}\}$. On définit I l'ensemble des indices $i \in \{1, \ldots, d-1\}$ tels que $q_i = p$. On rappelle que $1 \leq |\sigma_i| \leq |Q|$ pour chaque $i \in I$. Puisque $|I| \geq |Q| + 1$, il existe deux indices $i < j$ de I tels que $|\sigma_i| = |\sigma_j|$. Les mots σ_i et σ_j étiquettent, chacun, un circuit sur p. Ainsi, chaque mot de $\{\sigma_i, \sigma_j\}^*$ est un facteur d'un mot de $L(\mathcal{A})$. Puisque $L(\mathcal{A})$ est un langage borné, le lemme 5.5.4 de [Gin66, p. 171] montre que $\sigma_i \cdot \sigma_j = \sigma_j \cdot \sigma_i$. Il s'ensuit que $\sigma_i = \sigma_j$. Définissons maintenant $v = w_{i+1}\sigma_{i+1}^{n_{i+1}} \cdots w_{j-1}\sigma_{j-1}^{n_{j-1}} w_j$. Les mots v et σ_j étiquettent chacun un circuit sur p. Par l'argument précédent, on obtient que $v \cdot \sigma_j = \sigma_j \cdot v$. Il s'ensuit que

$$\begin{aligned} u &= w_1\sigma_1^{n_1} \cdots w_d\sigma_d^{n_d} \\ &= w_1\sigma_1^{n_1} \cdots w_i\sigma_i^{n_i} \cdot v \cdot \sigma_j^{n_j} \cdot w_{j+1}\sigma_{j+1}^{n_{j+1}} \cdots w_d\sigma_d^{n_d} \\ &= w_1\sigma_1^{n_1} \cdots w_i\sigma_i^{n_i} \cdot \sigma_j^{n_j} \cdot v \cdot w_{j+1}\sigma_{j+1}^{n_{j+1}} \cdots w_d\sigma_d^{n_d} \\ &= w_1\sigma_1^{n_1} \cdots w_i\sigma_i^{n_i+n_j} \cdot v \cdot w_{j+1}\sigma_{j+1}^{n_{j+1}} \cdots w_d\sigma_d^{n_d} \end{aligned}$$

Ainsi, $(q_0, w_1, q_1, \sigma_1, \ldots, w_{j-1}, q_{j-1}, \sigma_{j-1}, w_j w_{j+1}, q_{j+1}, \sigma_{j+1}, \ldots, w_d, q_d, \sigma_d)$ est une décomposition linéaire plate de w. Ceci est impossible, puisque le rang de cette décomposition est $(|w_1 \cdots w_d|, d-1)$, ce qui est strictement plus petit que le rang de la décomposition de rang minimal initiale.

On a montré que $d \leq |Q|^2+1$. On rappelle que $|w_i\sigma_i| \leq 2|Q|$ pour chaque indice $i \in \{1, \ldots, d\}$. Il s'ensuit que $d + |w_1\sigma_1 \cdots w_d\sigma_d| \leq d + 2d|Q| \leq 6|Q|^3$.

□

On considère maintenant une instance $(\mathcal{S}, \mathcal{A})$ du problème d'algébricité d'un système à compteurs plat. On dérive du lemme précédent que $\mathcal{L}(\mathcal{S}) \cap L(\mathcal{A})$ n'est pas algébrique si et seulement si il existe un calcul acceptant $q_0 \xrightarrow{w_1} q_1 \xrightarrow{\sigma_1} q_1 \cdots q_{d-1} \xrightarrow{w_d} q_d \xrightarrow{\sigma_d} q_d$ de $\mathcal{A}$ de longueur polynomiale tel que $\mathcal{L}(\mathcal{S}) \cap w_1\sigma_1^+ \cdots w_d\sigma_d^+$ n'est pas algébrique. Puisque l'algébricité de $\mathcal{L}(\mathcal{S}) \cap w_1\sigma_1^+ \cdots w_d\sigma_d^+$ peut être vérifiée en NP, on obtient que l'algébricité d'un système à compteurs plat est dans coNP.

La borne inférieure de la complexité est montrée par une réduction depuis 3-SAT. Considérons une instance de 3-SAT φ en forme normale conjonctive

$$\varphi \;=\; \bigwedge_{i=1}^{n} \underbrace{\ell_i^1 \vee \ell_i^2 \vee \ell_i^3}_{\chi_i}$$

où les ℓ_i^j sont des littéraux sur un ensemble $\{p_1, \ldots, p_m\}$ de variables booléennes. On construit, en temps linéaire, une instance $(c, \vec{c}_{\text{Init}}, \mathcal{A})$ du problème d'algébricité pour les systèmes d'addition de vecteurs tels que φ est satisfaisable si et seulement si le langage $\mathcal{L}(\mathcal{S}) \cap L(\mathcal{A})$ n'est pas algébrique. Soit $c = n + 2$. Informellement, les n premières composantes d'une configuration comptent le nombre de fois que chaque clause χ_i est satisfaite. Le deux dernières composantes sont utilisées pour générer des langages non-algébriques de la forme $\{\theta_1^{n_1}\theta_2^{n_2}\theta_3^{n_3} \mid n_1 \geq n_2, n_3\}$. La configuration initiale est $\vec{c}_{\text{Init}} = \vec{0}$, ce qui signifie que, initialement, aucune clause n'a encore été satisfaite. On choisit l'automate fini $\mathcal{A}$ qui reconnaît le langage $(\theta_{p_1} + \theta_{\neg p_1}) \ldots (\theta_{p_m} + \theta_{\neg p_m}).\theta.\theta_1^*\theta_2^*\theta_3^*$ où θ_ℓ est la transition de système d'addition de vecteurs qui incrémente chaque composante i satisfaisant $\ell \in \{\ell_i^1, \ell_i^2, \ell_i^3\}$. La transition de système d'addition de vecteurs θ décrémente les n premières composantes, la transition θ_1 incrémente les deux dernières composantes, les transitions θ_2 et θ_3 décrémentent respectivement les composantes $n+1$ et $n+2$. Remarquons que $\mathcal{L}(\mathcal{S}) \cap L(\mathcal{A})$ est non-vide si et seulement si φ est satisfaisable. De plus, dans ce cas l'intersection n'est pas algébrique puisque son image par le morphisme de mots envoyant toutes les transitions sur ε, excepté θ_1, θ_2 et θ_3 est le langage $\{\theta_1^{n_1}\theta_2^{n_2}\theta_3^{n_3} \mid n_1 \geq n_2, n_3\}$ qui est non-algébrique. Il s'ensuit que le problème de 3-SAT se réduit en temps polynomial au problème de non-algébricité pour les systèmes d'addition de vecteurs plats.

On en déduit que le problème d'algébricité pour les systèmes d'addition de vecteurs plats est coNP-complet.

Ceci conclut la preuve du théorème 5.23.

5.5.2 Algébricité des traces de systèmes d'addition de vecteurs

On souhaite maintenant s'intéresser à l'algébricité des traces de systèmes d'addition de vecteurs dans toutes leur généralité. On a déjà vu au paragraphe précédent, que si on se restreint au cas plat, ce problème est coNP-complet. On montre ici que ce problème reste décidable dans le cas des systèmes d'addition de vecteurs généraux, bien que la complexité soit supérieure.

Pour cela, on va montrer le théorème suivant :

Théorème 5.25. *Étant donné un système d'addition de vecteurs $\mathcal{V}$, $\mathcal{L}(\mathcal{V})$ est algébrique si et seulement si, pour tout langage L de la forme $\sigma_1^* \cdots \sigma_k^*$, $\mathcal{L}(\mathcal{V}) \cap L$ est algébrique.*

Dans le cas général, ce théorème est faux, comme le montre l'exemple suivant :

Exemple 5.11. *On considère le langage $L = \{u_1 \cdots u_n \mid n \geq 1\}$, avec pour tout n, $u_n = a^n b^n$. Ce langage n'est pas algébrique. Cependant, son intersection avec tout langage de la forme $\sigma_1^* \cdots \sigma_k^*$ est finie et donc algébrique.*

Ce langage est donc un contre-exemple à ce théorème dans le cas général. Remarquons qu'il n'est pas reconnaissable par un système d'addition de vecteurs.

On va montrer que dans le cas où un sytème d'addition de vecteurs a un langage de trace non-algébrique, on peut construire effectivement un langage de la forme $\sigma_1^* \cdots \sigma_k^*$ dont l'intersection avec le langage des traces n'est pas algébrique, qui sera appelé *témoin de non-algébricité*.

L'idée générale est de construire un automate à pile qui va tenter de simuler un système d'addition de vecteur. La procédure qui va tenter de construire cet automate à pile aura deux issues possible : soit elle réussit à construire un automate à pile qui simule exactement le système d'addition de vecteurs et reconnaît son langage de traces, soit il atteint en un temps fini un état d'échec, et le chemin qui y a mené donne un témoin de non-algébricité.

Comme on essaye de simuler le comportement d'un système d'addition de vecteurs par un automate à pile, la seule manière de stocker les vecteurs rencontrés est de les empiler sur la pile. Comme dans tout système de la sorte, il faut considérer les comportements asymptotiques, et donc s'occuper des vecteurs qu'on peut répéter un nombre arbitraire de fois. Un tel vecteur dont toutes les composantes sont positives ou nulles peut être lu sans restriction et peut donc être simplement empilé. Un tel vecteur dont au moins une des composantes est négative doit être compensé par un vecteur lu au préalable et donc présent sur la pile. On va donc commencer par définir le concept de *mariage* pour formaliser la compensation d'un vecteur ayant au moins une composante négative par un vecteur positif.

Mariages et mariages itérables On va définit le concept de *mariage* de deux vecteurs qui relie un vecteur positif et un vecteur non positif, et leur *ratio*, leur *somme d'appariement* et leur *reste* pour décrire ce qu'il se passe lorsque l'automate à pile qu'on va construire dépile le vecteur positif alors qu'il lit le vecteur non positif.

Intuitivement, le *ratio* d'un mariage $(\vec{v_1}, \vec{v_2})$ est le nombre de $\vec{v_2}$ que l'on peut lire en dépilant un $\vec{v_1}$, la somme d'appariement est le vecteur qu'il reste à compenser ou à rempiler après avoir compensé un $\vec{v_2}$ par un $\vec{v_1}$, et le reste est le vecteur des composantes que l'automate à pile ne peut pas garder en mémoire après cette compensation (car on pourrait compenser un nombre arbitraire de $\vec{v_2}$ par des $\vec{v_1}$, et avoir ainsi un reste arbitrairement grand qui ne pourrait être retenu qu'au sommet de la pile alors qu'il ne permet pas de lire le $\vec{v_2}$ suivant).

Définition 5.8. *Un couple de vecteurs* $(\vec{v_1}, \vec{v_2})$ *de* $\mathbb{Q}^d$ *est appelé* mariage *si* $\vec{v_1} \geq \vec{0}$ *et* $\vec{v_2} \not\geq \vec{0}$.
Il est appelé mariage itérable *si de plus,* $\forall i, \vec{v_2}(i) < 0 \Rightarrow \vec{v_1}(i) > 0$.

Proposition 5.26. *Pour tout mariage* (v_1, v_2)*, il existe un nombre rationnel positif maximal* λ *tel que* $\vec{v_1} + \lambda\vec{v_2} \geq \vec{0}$*. On appelle ce nombre le* ratio *du mariage et on le note* $\mathrm{rat}(v_1, v_2)$.
Remarquons qu'un mariage est itérable si et seulement si son ratio n'est pas nul.

Définition 5.9. *On définit la* somme d'appariement *du mariage* (v_1, v_2) *:*

$$\mathrm{mat}(\vec{v_1}, \vec{v_2}) = \begin{cases} (1-\lambda)\cdot\vec{v_2} & \text{if } \lambda < 1 \\ (1-\frac{1}{\lambda})\cdot\vec{v_1} & \text{if } \lambda \geq 1 \end{cases}$$

où $\lambda = \mathrm{rat}(\vec{v_1}, \vec{v_2})$.
On définit le reste *du mariage :* $\mathrm{rem}(\vec{v_1}, \vec{v_2}) = \vec{v_1} + \vec{v_2} - \mathrm{mat}(\vec{v_1}, \vec{v_2})$*. Ce reste est toujours un vecteur positif.*

Exemple 5.12. *On considère les vecteurs* $\vec{u} = (1, 2, 1)$ *et* $\vec{v} = (-1, -3, 2)$.
Le plus grand rationnel λ *tel que* $\vec{u} + \lambda\vec{v} \geq 0$ *est* $\lambda = \frac{2}{3}$*. On a donc* $\mathrm{rat}(\vec{u}, \vec{v}) = \frac{2}{3}$.
Après avoir compensé $\vec{v}$ *par* $\vec{u}$*, il reste* $\mathrm{mat}(\vec{u}, \vec{v}) = (1 - \mathrm{rat}(\vec{u}, \vec{v}))\cdot\vec{v} = (-\frac{1}{3}, -1, \frac{2}{3})$.
Le reste est $\mathsf{rem}(\vec{u}, \vec{v}) = \vec{u} + \lambda\vec{v} = (\frac{1}{3}, 0, \frac{7}{3})$.

Témoin de non-algébricité Dans ce paragraphe, on montre comment les mariages peuvent être utilisés pour obtenir un langage borné non-algébrique inclus dans le langage de traces d'un système d'addition de vecteurs témoignant de la non-algébricité de ce dernier.

L'idée est d'extraire un langage $\sigma_1^* \cdots \sigma_k^*$ et une relation bien parenthésée sur $\{1, k\}$ qui décrit quels sont les couples utilisés par l'automate à pile tentant de reconnaître $\mathcal{L}(\mathcal{V}) \cap \sigma_1^* \cdots \sigma_k^*$ tel que la somme de tous les restes des mariages correspondant à cette relation permette de lire σ_k. Cela signifie que l'automate à pile aura vidé sa pile alors que le système d'addition de vecteur est capable de lire une occurrence de σ_k de plus. Et que ce langage n'est pas algébrique.

Dans la suite, étant donné un mot $\sigma = \vec{v_1} \cdots \vec{v_n} \in \Theta^*$, on note $\Delta(\sigma) = \vec{v_1} + \cdots + \vec{v_n}$ le *déplacement* de σ.

Définition 5.10. *Étant donné une suite $(\sigma_1, \cdots, \sigma_k)$ de mots de Θ^*, un couple d'indices (s,t) est appelé mariage (resp. mariage itérable) si le couple de vecteurs $(\Delta(\sigma_s), \Delta(\sigma_t))$ est un mariage (resp. mariage itérable).*

Un schéma (resp. schéma itérable) est une relation bien parenthésée R sur $\{1, \cdots, k\}$ tel que toute couple $(s,t) \in R$ est mariage (resp. mariage itérable).

Le vecteur des excédents de R est $\mathrm{exc}(R) = \Sigma_{(s,t)\in R}\Delta(\sigma_s) + \mathrm{rat}(s,t)\Delta(\sigma_t)$.

Définition 5.11. *Un* témoin de non-algébricité *pour un système d'addition de vecteurs $\langle \Theta, \vec{c}_{\mathrm{Init}}\rangle$ est un tuple $(\sigma_1, \cdots, \sigma_k, U)$ où chaque $\sigma_j \in \Theta^*$ et U est un schéma tel que :*

$$\sigma_1 \cdots \sigma_k \text{ est une trace de } \langle \Theta, \vec{c}_{\mathrm{Init}}\rangle \tag{5.5.1}$$

$$\forall i, \Delta(\sigma_k)(i) < 0 \Rightarrow \mathrm{exc}(U)(i) > 0 \tag{5.5.2}$$

$$\text{Pour tout } (s,t) \in U \text{ avec } t < k, \text{ il existe un mariage itérable } (s',t) \in U \text{ avec } s' \leq s. \tag{5.5.3}$$

Exemple 5.13. *On considère le système d'addition de vecteurs $\mathcal{V} = \langle \Theta, \vec{c}_{\mathrm{Init}}\rangle$, avec :*

- $\Theta = \{\vec{a} = (-1,2), \vec{b} = (2,-1)\}$
- $\vec{c}_{\mathrm{Init}} = (1,1)$

Le tuple $(\vec{a}\vec{b}, \vec{a}, \vec{b}, \{(1,2)\})$ est un témoin de non-algébricité. En effet, $\{(1,2)\}$ est une relation bien parenthésée, et

- *$\vec{a}\vec{b}\vec{a}\vec{b}$ est une trace de $\mathcal{V}$.*
- *$\mathrm{exc}(U) = (0,3)$, donc $\|\mathrm{exc}(U)\|^+ = \|\vec{b}\|^-$.*
- *U contient un seul couple, qui est un mariage itérable, donc 5.5.3 est satisfaite.*

Proposition 5.27. *Si $\mathcal{V} = \langle \Theta, \vec{c}_{\mathrm{Init}}\rangle$ admet un témoin de non-algébricité, alors son langage de trace n'est pas algébrique.*

On montre cette proposition par contradiction. Prenons une suite $\sigma_1, \ldots, \sigma_k$ de mots de Θ^* telle que $\sigma_1 \cdots \sigma_k$ soit une trace à partir de $\vec{c}_{\mathrm{Init}}$, et supposons que $\mathcal{L}(\mathcal{V}) \cap \sigma_1^* \cdots \sigma_k^*$ est algébrique. On introduit l'ensemble $\vec{X}$ de vecteurs $(x_1, \ldots, x_k) \in \mathbb{Q}^k_{\geq 0}$ tel que $x_1\Delta(\sigma_1) + \cdots + x_j\Delta(\sigma_j) \geq \vec{0}$ pour tout $1 \leq j \leq k$. Tout vecteur de $\vec{X}$ peut être décomposé de manière bien parenthésée de la manière qui suit. Étant donné un mariage $r = (s,t)$, on introduit le vecteur $\vec{m}_r$ défini par :

$$\vec{m}_r = \vec{e}_s + \mathsf{rat}(s,t)\vec{e}_t$$

Remarquons que $\vec{m}_r \in \vec{X}$. Le lemme suivant procure une décomposition des vecteurs de $\vec{X}$ grâce aux schémas itérables.

Lemme 5.28. *Pour tout $\vec{x} \in \vec{X}$ il existe un schéma itérable R tel que :*

$$\vec{x} \in \sum_{j \mid \Delta(\sigma_j) \geq \vec{0}} \mathbb{Q}_{\geq 0}\vec{e}_j + \sum_{r \in R} \mathbb{Q}_{\geq 0}\vec{m}_r$$

Démonstration. Le théorème 5.3 nous assure que l'ensemble suivant est une union finie d'ensembles linéaires stratifiables :

$$\vec{N} = \{(n_1, \ldots, n_k) \in \mathbb{N}^k \mid \sigma_1^{n_1} \cdots \sigma_k^{n_k} \in \mathcal{L}(\vec{c}_{\mathrm{Init}})\}$$

On prend $\vec{x} = (x_1, \ldots, x_k)$ un vecteur de $\vec{X}$. Pour montrer le lemme, on suppose, sans perte de généralité, que $\vec{x} \in \mathbb{N}^k$. Puisque $\sigma_1 \cdots \sigma_k$ est une trace à partir de $\vec{c}_{\mathrm{Init}}$, on déduit, par monotonie, que pour tout $n \in \mathbb{N}$:

$$\sigma_1^{1+nx_1} \cdots \sigma_k^{1+nx_k}$$

est aussi une trace à partir de $\vec{c}_{\mathrm{Init}}$. Ainsi, $(1, \ldots, 1) + n\vec{x} \in \vec{N}$ pour tout $n \in \mathbb{N}$. Le principe des tiroirs nous donne l'existence d'un ensemble linéaire stratifiable qui contient $(1, \ldots, 1) + n\vec{x}$

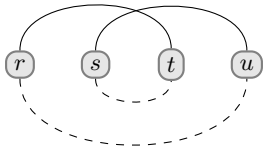


FIGURE 5.8 – Le lemme 5.29 suppose qu'il existe des arrêtes (r,t) et (s,u), représentées dans la partie haute, qui sont des mariages itérables. Dans ce cas, les arrêtes (r,u) et (s,t) sont des mariages itérables, représentés dans la partie basse.

pour une infinité de $n \in \mathbb{N}$ possibles. Il existe donc un ensemble périodique stratifiable $\vec{P}$ et un vecteur $\vec{b} = (b_1, \ldots, b_k) \in \mathbb{N}^k$, tels que $(1, \ldots, 1) + n\vec{x} \in (\vec{b} + \vec{P})$ pour une infinité de $n \in \mathbb{N}$. D'après le lemme de Dickson, on déduit que $\vec{x} \in \mathbb{Q}_{\geq 0}\vec{P}$.

Puisque $\vec{P}$ est stratifiable, il est généré par un ensemble fini stratifié $\vec{G}$. Prouvons que $\vec{G} \subseteq \vec{X}$. On prend un vecteur $\vec{g} = (g_1, \ldots, g_k)$ dans $\vec{G}$. Comme $\vec{b} + \mathbb{N}\vec{g} \subseteq \vec{N}$ on déduit que, pour tout $n \in \mathbb{N}$, le mot

$$\sigma_1^{b_1+ng_1} \cdots \sigma_k^{b_k+ng_k}$$

est une trace à partir $\vec{c}_{\text{Init}}$. Ainsi, $\vec{c}_{\text{Init}} + \Delta(\sigma_1^{b_1+ng_1} \cdots \sigma_j^{b_j+ng_j}) \geq \vec{0}$ pour tout $1 \leq j \leq k$ et $n \in \mathbb{N}$. Remarquons que ce vecteur est égal à $\vec{c}_{\text{Init}} + \Delta(\sigma_1^{b_1} \cdots \sigma_j^{b_j}) + n\vec{v}_j$ où $\vec{v}_j = g_1\Delta(\sigma_1) + \cdots + g_j\Delta(\sigma_j)$. On en déduit que $\vec{v}_j \geq -\frac{1}{n} \cdot (\vec{c}_{\text{Init}} + \Delta(\sigma_1^{b_1} \cdots \sigma_j^{b_j}))$. Ainsi $\vec{v}_j \geq \vec{0}$. On a prouvé que $\vec{g} \in \vec{X}$.

Étant donné que $\vec{G}$ est stratifié, il existe une relation bien parenthésée U sur $\{1, \ldots, k\}$ telle que $\vec{G} \subseteq \bigcup_{(s,t)\in U} \mathbb{Q}_{\geq 0}\vec{e}_s + \mathbb{Q}_{\geq 0}\vec{e}_t$. Soit R l'ensemble des mariages $(s,t) \in U$ qui sont itérable. On considère l'ensemble suivant :

$$\vec{C} = \sum_{j|\Delta(\sigma_j)\geq\vec{0}} \mathbb{Q}_{\geq 0}\vec{e}_j + \sum_{r\in R} \mathbb{Q}_{\geq 0}\vec{m}_r$$

Prouvons maintenant que $\vec{G} \subseteq \vec{C}$. On choisit $\vec{g} \in \vec{G}$. Il existe $(s,t) \in U$ tel que $\vec{g} \in \mathbb{Q}_{\geq 0}\vec{e}_s + \mathbb{Q}_{\geq 0}\vec{e}_t$. Supposons dans un premier temps que $s = t$. Puisque $\vec{g} \in \vec{X}$, on déduit que $g_s\Delta(\sigma_s) \geq \vec{0}$. Si $\Delta(\sigma_s) \geq \vec{0}$ alors $\vec{g} \in \vec{C}$, et si $\Delta(\sigma_s) \not\geq \vec{0}$ alors $g_s = 0$, ce qui entraîne que $\vec{g} = \vec{0} \in \vec{C}$. Ainsi, on obtient dans les deux cas $\vec{g} \in \vec{C}$. Supposons maintenant que $s < t$. Puisque $\vec{g} \in \vec{X}$ on déduit que $g_s\Delta(\sigma_s) \geq \vec{0}$ et $g_s\Delta(\sigma_s) + g_t\Delta(\sigma_t) \geq \vec{0}$. Si $g_s = 0$ ou $g_t = 0$, on déduit comme précédemment que $\vec{g} \in \vec{C}$. Ainsi, on peut supposer que $g_s > 0$ et $g_t > 0$. Du fait que $g_s\Delta(\sigma_s) \geq \vec{0}$, on obtient que $\Delta(\sigma_s) \geq \vec{0}$. Remarquons que si $\Delta(\sigma_t) \geq \vec{0}$ alors $\vec{g} \in \vec{C}$. On peut donc supposer que $\Delta(\sigma_t) \not\geq \vec{0}$. Dans ce cas $(\Delta(\sigma_s), \Delta(\sigma_t))$ est un mariage. Du fait que $g_s\Delta(\sigma_s) + g_t\Delta(\sigma_t) \geq \vec{0}$ on obtient $\Delta(\sigma_s) + \frac{g_t}{g_s}\Delta(\sigma_t) \geq \vec{0}$. Par maximalité du ratio, on a $\mathsf{rat}(s,t) \geq \frac{g_t}{g_s}$. En particulier, $\mathsf{rat}(s,t) > 0$. On déduit que (s,t) est un mariage itérable. Ainsi $(s,t) \in R$. L'égalité

$\vec{g} = (g_s - \frac{g_t}{\mathsf{rat}(s,t)})\vec{e}_s + \frac{g_t}{\mathsf{rat}(s,t)}(\vec{e}_s + \mathsf{rat}(s,t)\vec{e}_t)$ montre que $\vec{g} \in \vec{C}$.

Rappelons que $\vec{x} \in \mathbb{Q}_{\geq 0}\vec{P}$ et que $\vec{P}$ est généré par $\vec{G}$. On en déduit que $\vec{x}$ est une somme finie de vecteurs de $\mathbb{Q}_{\geq 0}\vec{G}$. Ainsi $\vec{x} \in \vec{C}$, puisque $\vec{G} \subseteq \vec{C}$, $\mathbb{Q}_{\geq 0}\vec{C} \subseteq \vec{C}$, et que $\vec{C}$ est stable par sommes finies. □

On déduit de ce théorème le lemme suivant, représenté en figure 5.8.

Lemme 5.29. *Pour tout $1 \leq r \leq s < t \leq u \leq k$ tel que (r,t) et (s,u) sont des mariages itérables, alors (r,u) et (s,t) sont des mariages itérables.*

Démonstration. Si $r = s$ ou $t = u$ le lemme est immédiat. On peut donc supposer, sans perte de généralité, que $r < s$ et $t < u$. On introduit le vecteur $\vec{x}$ défini par :

$$\vec{x} \ = \vec{m}_{r,t} \ + \ \vec{m}_{s,u} \tag{5.5.4}$$

Observons que $\vec{x} \in \vec{X}$. On déduit du lemme 5.28 qu'il existe un schéma itérable $R \subseteq \{r,s\}\times\{t,u\}$ tel que $\vec{x} \in \mathbb{Q}_{\geq 0}\vec{e}_r + \mathbb{Q}_{\geq 0}\vec{e}_s + \sum_{(i,j)\in R} \mathbb{Q}_{\geq 0}\vec{m}_{i,j}$. On obtient que $\vec{x}$ peut s'écrire comme

$$\vec{x} \;=\; \sum_{i\in\{r,s\}} \alpha_i \cdot \vec{e}_i \;+\; \sum_{(i,j)\in R} \alpha_{i,j} \cdot (\vec{e}_i + \mathsf{rat}(i,j)\cdot \vec{e}_j) \tag{5.5.5}$$

où les α_i et les $\alpha_{i,j}$ sont des rationnels non-négatifs tels que $\alpha_{i,j} = 0$ si $(i,j) \notin R$. On déduit de (5.5.4) et de (5.5.5) que les α_i et $\alpha_{i,j}$ satisfont le système d'équations suivant :

$$\begin{cases} \alpha_r + \alpha_{r,u} &= 1 - \alpha_{r,t} \\ \alpha_s + \alpha_{s,t} &= 1 - \alpha_{s,u} \\ \alpha_{s,t}\cdot \mathsf{rat}(s,t) &= (1-\alpha_{r,t})\cdot \mathsf{rat}(r,t) \\ \alpha_{r,u}\cdot \mathsf{rat}(r,u) &= (1-\alpha_{s,u})\cdot \mathsf{rat}(s,u) \end{cases}$$

Rappelons que $\mathsf{rat}(r,t) > 0$ et $\mathsf{rat}(s,u) > 0$ puisque (r,t) et (s,u) sont des mariages itérables. De plus, puisque R est bien parenthésée, $\alpha_{r,t} = 0$ ou $\alpha_{s,u} = 0$. On déduit du système d'équations précédent que $\mathsf{rat}(r,u) > 0$ et $\mathsf{rat}(s,t) > 0$. En conséquence, les couples (r,u) et (s,t) sont des mariages itérables. □

Supposons maintenant, par contradiction, qu'il existe un schéma U tel que $(\sigma_1,\ldots,\sigma_k,U)$ est un témoin de non-algébricité pour $\langle \Theta, \vec{c}_{\mathrm{Init}}\rangle$. Dans ce cas il existe $\mu > 0$ tel que $\mathrm{exc}(U)+\mu\Delta(\sigma_k) \geq \vec{0}$. On introduit le vecteur $\vec{x}$ défini par :

$$\vec{x} = \mu\vec{e}_k + \sum_{u\in U} \vec{m}_u \tag{5.5.6}$$

Observons que $\vec{x} \in \vec{X}$. On déduit du lemme 5.28 qu'il existe un schéma itérable R tel que :

$$\vec{x} \in \sum_{j\mid \Delta(\sigma_j)\geq\vec{0}} \mathbb{Q}_{\geq 0}\vec{e}_j + \sum_{r\in R} \mathbb{Q}_{\geq 0}\vec{m}_r \tag{5.5.7}$$

On déduit de (5.5.6) et (5.5.7) qu'il existe $\vec{z} \in \sum_{j\mid\Delta(\sigma_j)\geq\vec{0}} \mathbb{Q}_{\geq 0}\vec{e}_j$ tel que :

$$\mu\vec{e}_k + \sum_{u\in U} \vec{m}_u \quad\in\quad \vec{z} + \sum_{r\in R} \mathbb{Q}_{\geq 0}\vec{m}_r$$

En retirant de l'équation précédente les vecteurs $\vec{m}_{s,t}$ apparaissant des deux côtés, on construit deux suites $(\alpha_u)_{u\in U}$ et $(\beta_r)_{r\in R}$ de rationnels non-négatifs tels que $\alpha_{s,t}\beta_{s,t} = 0$ pour chaque $(s,t) \in U \cap R$ et tels que :

$$\mu\vec{e}_k + \sum_{u\in U} \alpha_u\cdot\vec{m}_u \quad=\quad \vec{z} + \sum_{r\in R} \beta_r\cdot\vec{m}_r \tag{5.5.8}$$

On considère la relation $\tilde{U}$ constituée des mariages itérables $u \in U$ tels que $\alpha_u > 0$ et l'ensemble $\tilde{R}$ de couples $r \in R$ tels que $\beta_r > 0$ (ces couples sont des mariages itérables puisque R est un schéma itérable). On a besoin des deux lemmes suivants, représentés en figure 5.9.

Lemme 5.30. *Il existe $1 \leq s < t' < t \leq k$ tels que $(s,t) \in \tilde{R}$ et $(s,t') \in \tilde{U}$.*

Démonstration. On considère le cas où $t = k$. Puisque $\mu > 0$, on déduit de (5.5.8) qu'il existe $r \in R$ tel que $\beta_r > 0$ et $\vec{m}_r(k) > 0$. Ainsi, il existe $s < k$ tel que $r = (s,k)$. Remarquons que $(s,k) \in \tilde{R}$. Puisque $\beta_r > 0$ et $\vec{m}_r(s) > 0$, on déduit de (5.5.8) qu'il existe $u \in U$ tel que $\alpha_u > 0$ et $\vec{m}_u(s) > 0$. Il existe $t' > s$ tel que $u = (s,t')$. Remarquons que $t' = k$ implique que $u = r$ et $\alpha_u\beta_r > 0$ ce qui est impossible. Ainsi on a $t' < k$. Puisque U est un témoin de non-algébricité, on déduit de (5.5.3) qu'il existe $s_0 \leq s$ tel que (s_0,t') est un mariage itérable. Puisque $s_0 \leq s < t' < t$ et que (s_0,t') et (s,t) sont des mariages itérables, le lemme 5.29 montre que (s,t') est un mariage itérable. Ainsi $(s,t') \in \tilde{U}$. □

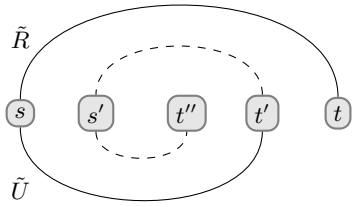


FIGURE 5.9 – Les mariages itérables (s', t') et (s', t'') construits dans la preuve du lemme 5.31. Les arrêtes représentent les couples $(s, t), (s', t') \in \tilde{R}$ pour la moitié haute, et $(s, t'), (s', t'') \in \tilde{U}$ pour la moitié basse.

Lemme 5.31. *Pour chaque $1 \leq s < t' < t \leq k$ tel que $(s, t) \in \tilde{R}$ et $(s, t') \in \tilde{U}$, il existe $s < s' < t'' < t'$ tel que $(s', t') \in \tilde{R}$ et $(s', t'') \in \tilde{U}$.*

Démonstration. Comme $(s, t') \in \tilde{U}$, on a que $\alpha_{s,t'} > 0$ et que $\vec{m}_{s,t'}(t') > 0$. Comme $\Delta(\sigma_{t'}) \not\geq \vec{0}$ on obtient que $\vec{z}(t') = 0$. (5.5.8) nous permet de déduire l'existence de $r \in R$ tel que $\beta_r > 0$ et $\vec{m}_r(t') > 0$. Ainsi $r \in \tilde{R}$ et il existe $s' < t'$ tel que $r = (s', t')$. Puisque R est bien parenthésé et que $(s, t) \in R$ on déduit que $s \leq s'$. Remarquons que puisque $s = s'$, on obtient que $\beta_r = \beta_{s,t'} > 0$, ce qui est en contradiction avec $\alpha_{s,t'}\beta_{s,t'} = 0$. On a ainsi $s < s' < t' < t$. Puisque $\Delta(\sigma_{s'}) \geq \vec{0}$ on obtient $\mu\vec{e}_k(s') = 0$. Comme $\beta_r > 0$ et $\vec{m}_r(s') > 0$, on déduit de (5.5.8) l'existence de $u \in U$ tel que $\alpha_u > 0$ et $\vec{m}_u(s') > 0$. Ainsi, il existe $t'' > s'$ tel que $u = (s', t'')$. Remarquons que si $t' < t''$, alors $s < s' < t' < t''$ ce qui est impossible puisque $(s, t'), (s', t'') \in U$ et que U est bien parenthésé. Ainsi $t'' \leq t'$. Remarquons que le fait que $t'' = t'$ implique que $\alpha_u = \alpha_{s',t'} > 0$ ce qui est en contradiction avec le fait que $\alpha_{s',t'}\beta_{s',t'} = 0$. On a donc $t'' < t'$. En particulier $t'' < k$ et puisque U est un témoin de non-algébricité, on déduit de (5.5.3) l'existence de $s_0 \leq s'$ tel que (s_0, t'') est un mariage itérable. Puisque (s_0, t'') et (s', t') sont des mariages itérables et que $s_0 \leq s' < t'' < t'$, le lemme 5.29 montre que (s', t'') est un mariage itérable. Ainsi $(s', t'') \in \tilde{U}$. □

On déduit des deux lemmes précédents l'existence d'une suite infinie $(s_i, t_i)_{i \geq 1}$ telle que $1 \leq s_i < t_{i+1} < t_i \leq k$, $(s_i, t_i) \in \tilde{R}$, et $(s_i, t_{i+1}) \in \tilde{U}$. Informellement, cela signifie que le phénomène représenté sur la figure 5.9 peut être itéré pour construire une spirale infinie. La contradiction est fournie par la finitude de l'ensemble $\{1, \ldots, k\}$. Ce qui conclut la preuve de la proposition 5.27.

On a donc que si on peut extraire un témoin de non-algébricité d'un système d'addition de vecteur, son langage de trace n'est pas algébrique. Il nous reste à montrer que si le langage de trace n'est pas algébrique, un tel témoin existe toujours. Pour cela, on va construire un *automate à pile de vecteurs* fini simulant le système d'addition de vecteur, et lorsque le langage est algébrique, cet automate à pile de vecteurs simule fidèlement ce système d'addition de vecteur, et que dans le cas contraire, il atteint un état spécial d'échec via un témoin de non-algébricité. Le reste du paragraphe est consacrée à présenter ce modèle d'automate à pile de vecteurs, montrer que dans le cas où un automate à pile est fini, il est équivalent à un automate à pile, et finalement montrer qu'étant donné un système d'addition de vecteurs, on peut construire un automate à pile de vecteurs fini qui soit reconnaît le langage des traces du système d'addition de vecteurs, soit permet d'exhiber un témoin de non-algébricité.

Automate à pile de vecteurs On introduit maintenant une extension des automates à pile dans le but de simuler le comportement d'un système d'addition de vecteurs. Informellement, un automate à pile de vecteur est un automate fini équipé de deux mémoires non bornées : un compteur `r` contenant un vecteur de $\mathbb{Q}^d_{\geq 0}$, et une pile `z`, où chaque symbole est un vecteur

de $\mathbb{Q}^d_{\geq 0}$. Les actions sur le compteurs sont limitées aux translations par un vecteur de $\mathbb{Q}^d_{\geq 0}$, et les actions sur la pile sont les actions de push et de pop habituelles. De plus, l'automate peut tester la satisfaction de contraintes linéaires, avec des coefficients négatifs, mettant en jeu son compteur et la somme de tous les vecteurs empilés. Formellement, on définit l'ensemble des opérations $\mathtt{Op} = \mathtt{Op}_{cnt} \cup \mathtt{Op}_{lifo} \cup \mathtt{Op}_{test}$ par

$$\begin{aligned} \mathtt{Op}_{cnt} &= \{\mathsf{add}(\vec{v}) \mid \vec{v} \in \mathbb{Q}^d_{\geq 0}\} \\ \mathtt{Op}_{lifo} &= \{\mathrm{push}_{\vec{\gamma}}, \mathrm{pop}_{\vec{\gamma}} \mid \vec{\gamma} \in \mathbb{Q}^d_{\geq 0}\} \\ \mathtt{Op}_{test} &= \{\mathsf{test}(\varphi) \mid \varphi \in \Phi\} \end{aligned}$$

ou Φ est l'ensemble de toutes les combinaisons booléennes de contraintes de la forme

$$\sum_{i=1}^{d} \alpha_i \mathtt{r}(i) + \sum_{i=1}^{d} \beta_i \Delta(\mathtt{z})(i) \ \# \ c$$

avec $\alpha_i, \beta_i \in \mathbb{N}$, $\# \in \{\leq, \geq\}$, et $c \in \mathbb{Z}$. On sous-entend dans la suite qu'une formule $\varphi \in \Phi$ a $\mathtt{r}$ et $\mathtt{z}$ comme variables libres.

Un *automate à pile de vecteurs* est un quintuplet $\mathcal{P} = \langle Q, q_{\mathrm{Init}}, \Sigma, T\rangle$ où Q est un ensemble d'*états*, $q_{\mathrm{Init}} \in Q$ est l'état *initial*, Σ est l'*alphabet d'entrée*, et $T \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathtt{Op} \times Q$ est un ensemble de *transitions*. Il n'y a pas d'ensemble d'états finaux explicite, étant donné que nous n'en avons pas besoin. Remarquons également que les ensembles Q et T peuvent être infinis. Une transition $(q, \ell, \mathtt{op}, q')$, également écrite $q \xrightarrow[\ell]{\mathtt{op}} q'$, signifie que l'automate va de l'état q à l'état q' en lisant la lettre d'entrée $\ell \in (\Sigma \cup \{\varepsilon\})$ et en réalisant l'opération $\mathtt{op}$. ℓ est l'*étiquette* de t, et $\mathtt{op}$ son *opération*.

On donne la sémantique opérationnelle de $\mathcal{P}$ comme un système de transitions étiqueté. L'ensemble des *configurations* est $\mathcal{C} = Q \times \mathbb{Q}^d_{\geq 0} \times (\mathbb{Q}^d_{\geq 0})^*$, et la *configuration initiale* est $(q_{\mathrm{Init}}, \vec{0}, \varepsilon)$. La *relation* définie par une transition $t = (q, \ell, \mathtt{op}, q')$, écrite $\xrightarrow{t}$, est la plus petite relation binaire sur $\mathcal{C}$ satisfaisant les conditions suivantes :

$$\begin{aligned} (q, \vec{r}, z) &\xrightarrow{t} (q', \vec{r} + \vec{v}, z) && \text{si} \quad \mathtt{op} = \mathsf{add}(\vec{v}) \\ (q, \vec{r}, z) &\xrightarrow{t} (q', \vec{r}, z\vec{\gamma}) && \text{si} \quad \mathtt{op} = \mathrm{push}_{\vec{\gamma}} \\ (q, \vec{r}, z\vec{\gamma}) &\xrightarrow{t} (q', \vec{r}, z) && \text{si} \quad \mathtt{op} = \mathrm{pop}_{\vec{\gamma}} \\ (q, \vec{r}, z) &\xrightarrow{t} (q', \vec{r}, z) && \text{si} \quad \mathtt{op} = \mathsf{test}(\varphi) \text{ et } \vec{r}, z \models \varphi \end{aligned}$$

Une *exécution* dans $\mathcal{P}$ est une suite alternée finie $(c_0, t_1, c_1, \ldots, t_k, c_k)$ de configurations $c_i \in \mathcal{C}$ et de transitions $t_i \in T$, satisfaisant $c_{i-1} \xrightarrow{t_i} c_i$ pour tout i. L'*étiquette* d'une exécution est le mot $\ell_1 \cdots \ell_k \in \Sigma^*$, où chaque ℓ_i est l'étiquette de t_i. Par soucis de brièveté, on remplacera parfois une transition t par son étiquette et/ou son opération lorsque l'on parle de pas et d'exécutions. Le *langage* reconnu par $\mathcal{P}$ est l'ensemble des mots de Σ^* qui étiquettent une exécution à partir de la configuration initiale. Un état (resp. transition, configuration) est appelée *atteignable* dans $\mathcal{P}$ quand elle apparaît dans une exécution démarrant dans la configuration initiale.

Théorème 5.32. *Si un automate à pile de vecteur a un ensemble fini de transitions, alors c'est un automate à pile.*

Démonstration. Soit $\mathcal{P} = \langle Q, q_{\mathrm{Init}}, \Sigma, T\rangle$ un automate à pile de vecteurs dont on suppose que Q et T sont finis Remarquons que le langage reconnu par $\mathcal{P}$ est préservé quand tous les vecteurs apparaissant dans les opérations de T sont multipliés par le même entier positif. On peut donc supposer sans perte de généralité que tout vecteur apparaissant dans les opérations de T sont des vecteurs d'entiers. Pour montrer que le langage reconnu par $\mathcal{P}$ est algébrique, on construit

à partir de $\mathcal{P}$ un nouvel automate à pile de vecteur $\mathcal{Q}$ sons opérations $\mathsf{add}(\vec{v})$ ni $\mathsf{test}(\varphi)$. On définit K la constante maximale c apparaissant dans les opérations $\mathsf{test}(\varphi)$ de T. La constante K agit comme un seuil pour abstraire les grandes composantes du compteur et de la somme des vecteurs empilés. Formellement, l'*abstraction* d'un vecteur $\vec{x} \in \mathbb{N}^d$ est le vecteur $\vec{x}^\sharp$ dans $(\{0, \ldots, K\} \cup \{\star\})^d$, défini par $\vec{x}^\sharp(i) = \star$ si $\vec{x}(i) > K$ et $\vec{x}^\sharp(i) = \vec{x}(i)$ sinon. Le compteur de $\mathcal{P}$ est remplacé par son abstraction dans $\mathcal{Q}$. Ce remplacement est possible car toutes les opérations font croître le compteur. Comme le compteur abstrait de $\mathcal{Q}$ peut prendre uniquement un nombre fini de valeurs, on le stocke comme une partie de l'état. On maintient également la somme abstraite de la somme de tous les vecteurs empilés dans $\mathcal{Q}$. Pour cela, $\mathcal{Q}$ code un contenu de pile $\vec{\gamma}_1 \cdots \vec{\gamma}_h$ de $\mathcal{P}$ par $\vec{\gamma}_1 \vec{s}_1 \cdots \vec{\gamma}_h \vec{s}_h$, où chaque $\vec{s}_i$ satisfait $\vec{s}_i = (\sum_{j=1}^{i} \vec{\gamma}_j)^\sharp$. Remarquons que $\mathcal{Q}$ a seulement besoin d'un nombre fini d'états et de transitions supplémentaires pour stocker cette encodage. En effet, l'information procurée par la vecteur $\vec{s}$ au sommet de la pile (ainsi que le compteur abstrait) est suffisante pour simuler les opérations $\mathsf{test}(\varphi)$ de $\mathcal{P}$. On en déduit que $\mathcal{P}$ et $\mathcal{Q}$ reconnaissent le même langage. Puisque $\mathcal{Q}$ contient seulement des opérations de piles de $\mathtt{Op}_{lifo}$ et a un nombre fini d'états et de transitions, le langage qu'il reconnaît est algébrique. □

Simuler un système d'addition de vecteurs par un automate à pile de vecteurs Dans tout ce paragraphe, on fixe un système d'addition de vecteurs $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$. On appelle *ensemble support* tout sous-ensemble W de Θ^* qui est non-vide et clos par préfixe. On introduit un automate à pile de vecteurs $\mathcal{P}_W$ qui simule le comportement de $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$. La simulation est paramétrée par un ensemble support W. Comme il sera clarifié plus tard, l'ensemble support W a pour effet de restreindre le comportement de $\mathcal{P}_W$. En particulier, on montrera que $\mathcal{P}_W$ a un nombre fini d'états et de transitions atteignables si W est fini. Toutefois, il n'est pas nécessaire que W soit fini dans le cas général.

Formellement, étant donné un ensemble support $W \subseteq \Theta^*$, l'automate à pile de vecteurs $\mathcal{P}_W$, avec Θ comme alphabet d'entrée, est défini comme suit. Les états de $\mathcal{P}_W$ sont les couples $q = (\vec{p}, w)$ où $\vec{p}$ est un vecteur de $\mathbb{Q}^d$ et w est un mot de $W \cup W\Theta$. De plus, on introduit deux états puits $\bot$ et $\lightning$ pour représenter les échecs de la simulation. L'état initial est $(\vec{0}, \varepsilon)$. Les transitions de $\mathcal{P}_W$ seront formellement définies un peu plus loin, mais expliquons tout d'abord informellement le comportement. Il y a deux modes opératoires dépendant de la première composante de l'état : $\vec{p}$. Si $\vec{p} = \vec{0}$, alors $\mathcal{P}_W$ est *au repos*, c.à.d., prêt à lire un symbole d'entrée. Dans le cas contraire, $\mathcal{P}_W$ *traite* le vecteur $\vec{p}$. Décrivons une exécution de $\mathcal{P}_W$ sur un mot d'entrée u qui est une trace de $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$. Initialement, $\mathcal{P}_W$ est dans l'état $(\vec{0}, \varepsilon)$, avec un compteur à zéro et une pile vide. Premièrement, $\mathcal{P}_W$ lit des symboles d'entrée et les concatène au support w qui est une partie de l'état. Dès que le support se termine par un circuit [5] $\sigma \neq \varepsilon$ tel que $\Delta(\sigma) \geq \vec{0}$, $\mathcal{P}_W$ extrait le circuit σ du support, empile $\Delta(\sigma)$ sur la pile, et va dans l'état $(\vec{0}, w')$ où $w = w'\sigma$. Après cela, $\mathcal{P}_W$ reprend son calcul : il concatène l'entrée au support et en extrait les circuits. Toutefois, l'extraction de circuits est réalisée à l'aide de la pile : la condition $\Delta(\sigma) \geq \vec{0}$ devient $\vec{s} + \Delta(\sigma) \geq \vec{0}$, où $\vec{s}$ est la somme de tous les circuits précédemment extraits. Quand $\mathcal{P}_W$ extrait un circuit σ avec $\Delta(\sigma) \not\geq \vec{0}$, il se déplace dans l'état de traitement $(\Delta(\sigma), w')$ au lieu d'empiler $\Delta(\sigma)$ sur la pile. Alors, on dépile des vecteurs de la pile pour compenser $\Delta(\sigma)$. À chaque compensation, le reste est ajouté au compteur de $\mathcal{P}_W$. Si $\Delta(\sigma)$ est entièrement compensé par la pile, $\mathcal{P}_W$ se déplace dans l'état de repos $(\vec{0}, w')$. Si, au contraire, la pile est vidée avant que $\Delta(\sigma)$ ne soit entièrement traité, $\mathcal{P}_W$ se déplace dans l'état d'échec $\lightning$. La simulation échoue également quand la composante w de l'état sort de l'ensemble support W (après avoir lu un symbole d'entrée). Dans ce cas, $\mathcal{P}_W$ se déplace dans l'état d'échec $\bot$.

On précise maintenant les idées précédentes. Pour simplifier leur présentation, on utilisera quelques raccourcis de notation pour les formules φ utilisées dons les opérations $\mathsf{test}(\varphi)$. Étant

5. Dans notre configuration, un circuit n'est rien d'autre qu'une suite non-vide d'actions de Θ^*. On utilise le terme de circuit puisque puisque les suffixes extraits par $\mathcal{P}_W$ seront des circuits pour un système d'addition de vecteurs avec circuits.

donné un vecteur $\vec{v} \in \mathbb{Z}^d$ et $\# \in \{\leq, \geq\}$,

$$\mathtt{r} + \Delta(\mathtt{z}) + \vec{v} \,\#\, \vec{0} \quad \text{remplacera} \quad \bigwedge_{i=1}^{d} \mathtt{r}(i) + \Delta(\mathtt{z})(i) \,\#\, -\vec{v}(i)$$
$$\|\vec{v}\|^+ \subseteq \|\mathtt{r}\|^+ \quad \text{remplacera} \quad \bigwedge_{\vec{v}(i)>0} \neg(\mathtt{r}(i) \leq 0)$$

La formule $Extr(w)$ utilisée dans la première règle spécifie qu'on peut emprunter une transition «d'extraction de circuit» (cf troisième règle). Cette formule est définie par

$$Extr(w) \;=\; \bigvee_{\sigma \neq \varepsilon \text{ suffixe de } w} \mathtt{r} + \Delta(\mathtt{z}) + \Delta(\sigma) \;\geq\; \vec{0}$$

Formellement, l'ensemble (infini) des transitions de $\mathcal{P}_W$ est donné par les règles suivantes, où $\vec{p}$ est dans $\mathbb{Q}^d$, w dans W, et $\vec{\gamma}$ dans $\mathbb{Q}^d_{\geq 0}$.

· *Lecture d'un vecteur d'entrée.* Pour chaque vecteur $\vec{a} \in \Theta$,

$$(\vec{0}, w) \xrightarrow{\mathsf{test}(\neg Extr(w))} \cdot \xrightarrow[\vec{a}]{\mathsf{test}(\mathtt{r}+\Delta(\mathtt{z})+\vec{v} \geq \vec{0})} (\vec{0}, w\vec{a})$$

où $\vec{v} = \vec{c}_{\mathrm{Init}} + \Delta(w) + \vec{a}$.

· *Échec par sortie du support.* If $w \in W\Theta \setminus W$,

$$(\vec{0}, w) \to \bot$$

· *Extraction de circuit.* Pour tout mot $\sigma \in \Theta^*$ avec $w\sigma \in W$ et $\sigma \neq \varepsilon$,

$$(\vec{0}, w\sigma) \xrightarrow{\mathsf{test}(\mathtt{r}+\Delta(\mathtt{z})+\Delta(\sigma) \geq \vec{0})} (\Delta(\sigma), w)$$

· *Traitement d'un vecteur positif.* Si $\vec{p} > \vec{0}$,

$$(\vec{p}, w) \xrightarrow{\mathsf{test}(\|\vec{p}\|^+ \subseteq \|\mathtt{r}\|^+)} \cdot \xrightarrow{\mathsf{add}(\vec{p})} (\vec{0}, w)$$

$$(\vec{p}, w) \xrightarrow{\mathsf{test}(\|\vec{p}\|^+ \not\subseteq \|\mathtt{r}\|^+)} \cdot \xrightarrow{\mathrm{push}_{\vec{p}}} (\vec{0}, w)$$

· *Traitement d'un vecteur non-positif.* Si $\vec{p} \not\geq \vec{0}$,

$$(\vec{p}, w) \xrightarrow{\mathrm{pop}_{\vec{\gamma}}} \cdot \xrightarrow{\mathsf{add}(\mathsf{rem}(\vec{\gamma},\vec{p}))} (\mathrm{mat}(\vec{\gamma}, \vec{p}), w)$$

· *Échec par pile vide.* Si $\vec{p} \not\geq \vec{0}$,

$$(\vec{p}, w) \xrightarrow{\mathsf{test}(\Delta(\mathtt{z}) = \vec{0})} \lightning$$

L'automate à pile de vecteurs $\mathcal{P}_W$ pourrait sembler inutile à première vue, puisqu'il possède une infinité d'état et un alphabet de pile infini. En réalité, il constitue le principal ingrédient de notre algorithme (cf figure 5.10 page 120) résolvant le problème de l'algébricité d'un système d'addition de vecteurs. Avant cela, il est nécessaire d'établir quelques résultats préalables.

Montrons premièrement que $\mathcal{P}_W$ simule fidèlement le système d'addition de vecteurs $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$ quand il n'échoue pas. La notation suivante sera utile pour présenter formellement la simulation de $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$ par $\mathcal{P}_W$. Étant donné une configuration $(\vec{p}, w, \vec{r}, z)$ de $\mathcal{P}_W$, on note $\mathsf{val}(\vec{p}, w, \vec{r}, z)$ sa *valeur*, définie par

$$\mathsf{val}(\vec{p}, w, \vec{r}, z) \;=\; \vec{c}_{\mathrm{Init}} + \vec{p} + \Delta(w) + \vec{r} + \Delta(z) \tag{5.5.9}$$

Remarquons que la valeur reste constante lors de l'application de transitions de la forme «extraction de circuit» ou «traitement». Il s'ensuit, par une induction classique, que

$$\mathsf{val}(\vec{p}, w, \vec{r}, z) \;=\; \vec{c}_{\mathrm{Init}} + \Delta(u) \;\geq\; \vec{0} \tag{5.5.10}$$

pour toute exécution $(\vec{0}, \varepsilon, \vec{0}, \varepsilon) \xrightarrow[u]{*} (\vec{p}, w, \vec{r}, z)$ de $\mathcal{P}$. Ainsi, le langage reconnu par $\mathcal{P}_W$ est inclus dans la trace de $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$. L'inclusion inverse est fausse dans le cas général. Cependant, on donne une condition suffisante pour qu'elle soit vraie. Formellement, on dit que $\mathcal{P}_W$ *réussit* si ni $\bot$ ni $\lightning$ ne sont atteignables dans $\mathcal{P}_W$.

Lemme 5.33. *Si $\mathcal{P}_W$ réussit, alors il contient, pour chaque configuration accessible $(\vec{0}, w, \vec{r}, z)$, une exécution $(\vec{0}, w, \vec{r}, z) \xrightarrow[\varepsilon]{*} (\vec{0}, w', \vec{r}', z')$ telle que $\vec{r}', \Delta(z') \models \neg \mathit{Extr}(w')$.*

Démonstration. On considère une configuration accessible $(\vec{0}, w, \vec{r}, z)$. On montre que si $\vec{r}, \Delta(z) \models \mathit{Extr}(w)$, alors $\mathcal{P}_W$ peut atteindre, à partir de la configuration $(\vec{0}, w, \vec{r}, z)$, une configuration avec un état $(\vec{0}, w')$ où w' est un préfixe stricte de w. On obtient le lemme par une induction sur $|w|$.

Supposons que $\vec{r}, \Delta(z) \models \mathit{Extr}(w)$. Il existe un suffixe non-vide σ de w tel que $\vec{r} + \Delta(z) + \Delta(\sigma) \geq \vec{0}$. De plus, $w \in W$ puisque $\bot$ n'est pas atteignable dans $\mathcal{P}_W$. Ainsi, $\mathcal{P}_W$ peut, via une transition d'«extraction de circuit», se rendre dans une configuration avec un état $(\vec{p}', w')$, où $w = w'\sigma$. Si $\vec{p}' \not\geq \vec{0}$, alors $\mathcal{P}_W$ peut emprunter des transitions de «traitement d'un vecteur non-positif» et atteindre une configuration avec un état $(\vec{p}'', w')$ avec $\vec{p}'' \geq \vec{0}$. En effet, la pile ne peut être vidée dans le traitement, puisque $\mathcal{P}_W$ réussit. Ainsi, on peut supposer sans perte de généralité que $\vec{p}' \geq \vec{0}$. Si $\vec{p}' = \vec{0}$ la preuve est terminée. Dans le cas contraire, $\mathcal{P}_W$ peut emprunter une transition de «traitement d'un vecteur positif» et atteindre une configuration avec un état $(\vec{0}, w')$. □

Proposition 5.34. *Si $\mathcal{P}_W$ réussit, il reconnaît le langage des traces de $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$.*

Démonstration. Il reste à montrer que le langage reconnu par $\mathcal{P}_W$ contient le langage des traces de $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$. On montre par induction sur $|u|$ que, pour toute trace u de $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$, il existe une exécution $(\vec{0}, \varepsilon, \vec{0}, \varepsilon) \xrightarrow[u]{*} (\vec{0}, w, \vec{r}, z)$ dans $\mathcal{P}$. Le cas de base $u = \varepsilon$ est trivial. Pour montrer l'étape d'induction, considérons une trace $u\vec{a}$ of $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$, et supposons que $(\vec{0}, \varepsilon, \vec{0}, \varepsilon) \xrightarrow[u]{*} (\vec{0}, w, \vec{r}, z)$ est une exécution de $\mathcal{P}$. Puisque $\mathcal{P}_W$ réussit, on déduit du lemme 5.33 que $\mathcal{P}_W$ contient une exécution $(\vec{0}, w, \vec{r}, z) \xrightarrow[\varepsilon]{*} (\vec{0}, w', \vec{r}', z')$ telle que $\vec{r}', \Delta(z') \models \neg \mathit{Extr}(w')$. Remarquons que

$$\begin{aligned} \vec{c}_{\mathrm{Init}} + \Delta(w') + \vec{r}' + \Delta(z') + \vec{a} \;&=\; \mathsf{val}(\vec{0}, w', \vec{r}', z') + \vec{a} \\ &=\; \vec{c}_{\mathrm{Init}} + \Delta(u) + \vec{a} \\ &\geq\; \vec{0} \end{aligned}$$

On en déduit que $\mathcal{P}_W$ contient une transition de «lecture» $(\vec{0}, w', \vec{r}', z') \xrightarrow[a]{t} (\vec{0}, w'\vec{a}, \vec{r}', z')$, ce qui conclut la preuve. □

Montrons maintenant que si $\mathcal{P}_W$ atteint l'état $\lightning$ alors $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$ admet un témoin de non-algébricité. On obtient ce témoin à partir de la suite de circuits $(\sigma_1, \ldots, \sigma_k)$ extraits le long de l'exécution et la relation bien parenthésée correspond à l'appariement des opérations push/pop.

Une suite $(\sigma_1, \ldots, \sigma_k)$ de mots de Θ^* est dite *compensable* si $\|\Delta(\sigma_j)\|^- \subseteq \|\Delta(\sigma_1)\|^+ \cup \cdots \cup \|\Delta(\sigma_{j-1})\|^+$ pour tout $1 \leq j \leq k$. Dans chaque cas, l'ensemble d'indices $I = \|\Delta(\sigma_1)\|^+ \cup \cdots \cup \|\Delta(\sigma_k)\|^+$ est appelée l'ensemble des composantes *augmentées*.

Lemme 5.35. *Pour toute suite compensable $(\sigma_1, \ldots, \sigma_k)$ il existe une suite $n_1, \ldots, n_k \in \mathbb{N}$ telle que le vecteur $\vec{v}_j = \Delta(\sigma_1^{n_1} \cdots \sigma_j^{n_j})$ est dans $\mathbb{N}^d$ pour chaque $1 \leq j \leq k$ et telle que $\|\vec{v}_k\|^+$ est l'ensemble des composantes augmentées.*

Démonstration. On montre le lemme par induction sur k. Le cas $k = 0$ est immédiat. Supposons que la propriété est démontrée pour $k \in \mathbb{N}$ et considérons une suite compensable $(\sigma_1, \ldots, \sigma_{k+1})$. Par induction, il existe $n_1, \ldots, n_k \in \mathbb{N}$ tel que le vecteur $\vec{v}_j = \Delta(\sigma_1^{n_1} \cdots \sigma_j^{n_j})$ est dans $\mathbb{N}^d$ pour chaque $1 \leq j \leq k$ et que $\|\vec{v}_k\|^+$ est l'ensemble des composantes positives de $(\sigma_1, \ldots, \sigma_k)$. Puisque $\|\Delta(\sigma_{k+1})\|^- \subseteq \|\Delta(\sigma_1)\|^+ \cup \cdots \cup \|\Delta(\sigma_k)\|^+ = \|\vec{v}_k\|^+$ on déduit qu'il existe $m \in \mathbb{N}$ telle que $m \cdot \vec{v}_k + \Delta(\sigma_{k+1}) \geq \vec{0}$. Considérons la suite $n'_1, \ldots, n'_{k+1}$ définie par $n'_j = (m+1) \cdot n_j$ si $j \leq k$ et $n'_{k+1} = 1$ et remarquons que cette suite prouve l'induction. □

Dans la suite, on montre que les circuits extraits le long d'un exécution de $\mathcal{P}_W$ sont compensables. On introduit l'ensemble $\mathbb{Q}_I^d$ de vecteurs $\vec{v} \in \mathbb{Q}^d$ tels que $\vec{v}(i) = 0$ pour chaque $i \notin I$, et l'ensemble Θ_I^* de mots $u \in \Theta^*$ tels que $\Delta(u) \in \mathbb{Q}_I^d$.

Lemme 5.36. *Considérons un vecteur $\vec{v} \in \mathbb{N}^d$ et un sous-ensemble $I \subseteq \|\vec{v}\|^+$. Si il existe une trace depuis une configuration $\vec{c}$ étiquetée par un mot de $\Theta_I^* \vec{a}_1 \cdots \Theta_I^* \vec{a}_n$ alors il existe $r \in \mathbb{N}$ et une trace depuis $\vec{c} + r \cdot \vec{v}$ étiquetée par $\vec{a}_1 \cdots \vec{a}_n$*

Démonstration. Considérons les mots $u_1, \ldots, u_n \in \Theta_I^*$ tels que $u_1 \vec{a}_1 \cdots u_n \vec{a}_n$ est une trace depuis $\vec{c}$. On considère la configuration $\vec{c}_j = \vec{c} + \Delta(u_1 \vec{a}_1 \cdots u_j \vec{a}_j)$. Puisque $I \subseteq \|\vec{v}\|^+$ il existe $r \in \mathbb{N}$ tel que le vecteur $\vec{x}_j = \vec{c} + r \cdot \vec{v} + \Delta(\vec{a}_1 \cdots \vec{a}_j)$ satisfasse $\vec{x}_j(i) \geq 0$ pour chaque $i \in I$ et pour chaque $0 \leq j \leq n$. Puisque $\vec{x}_j(i) = \vec{c}_j(i)$ pour chaque $i \notin I$ on obtient $\vec{x}_j \in \mathbb{N}^d$. Comme $\vec{x}_{j-1} \xrightarrow{\vec{a}_j} \vec{x}_j$ pour chaque $1 \leq j \leq k$, on a montré qu'il existe une trace depuis $\vec{c} + r \cdot \vec{v}$ étiquetée $\vec{a}_1 \cdots \vec{a}_n$. □

Une suite $(\sigma_1, \ldots, \sigma_k)$ est dite *insérable* dans une trace u depuis $\vec{c}_{\text{Init}}$ s'il existe une décomposition de u en $u = u_1 \cdots u_{k+1}$ et une suite $m_1, \ldots, m_k \in \mathbb{N}_{>0}$ telle que $u_1 \sigma_1^{m_1} \cdots u_k \sigma_k^{m_k} u_{k+1}$ est une trace depuis $\vec{c}_{\text{Init}}$.

Lemme 5.37. *Pour toute configuration $(\vec{p}, \vec{a}_1 \cdots \vec{a}_n, \vec{r}, z)$ accessible dans $\mathcal{P}_W$ par une exécution étiquetée par u, la suite $(\sigma_1, \ldots, \sigma_k)$ de circuits extraite lors de l'exécution est insérable dans u, et elle est compensable avec un ensemble I de composantes augmentées telle que $\vec{p}, \vec{r} \in \mathbb{Q}_I^d$, $z \in (\mathbb{Q}_I^d)^*$, et telle que :*

$$u \in \Theta_I^* \vec{a}_1 \cdots \Theta_I^* \vec{a}_n \Theta_I^*$$

Démonstration. On réalise la preuve par induction sur la longueur des exécutions de l'automate à pile de vecteurs. Pour l'exécution vide, la preuve est immédiate. Supposons qu'on atteint une configuration $(\vec{p}, w, \vec{r}, z)$ via une exécution étiquetée par u. On note $(\sigma_1, \ldots, \sigma_k)$ la suite de circuits extraite le long de l'exécution. Considérons une configuration $(\vec{p}', w', \vec{r}', z')$ atteignable en un pas depuis $(\vec{p}, w, \vec{r}, z)$ dans $\mathcal{P}_W$. On suppose que $(\vec{p}, w, \vec{r}, z)$ satisfait le lemme. Cela signifie que $(\sigma_1, \ldots, \sigma_k)$ est insérable dans u, et qu'il est compensable avec un ensemble I de composantes augmentées tel que $\vec{p}, \vec{r} \in \mathbb{Q}_I^d$, $z \in (\mathbb{Q}_I^d)^*$, et $u \in \Theta_I^* \vec{a}_1 \cdots \Theta_I^* \vec{a}_n \Theta_I^*$ où $w = \vec{a}_1 \cdots \vec{a}_n$.

On développe uniquement le cas de la transition «extraction d'un circuit», l'induction étant immédiate pour les autres transitions. Dans ce cas, w peut être décomposé en $w = w'\sigma$ où σ est un mot non-vide, $\vec{p}' = \Delta(\sigma)$, $\vec{r}' = \vec{r}$, $z' = z$, et $\Delta(\sigma) + \vec{r} + \Delta(z) \geq \vec{0}$.

Puisque $\vec{r}(i) = 0$ et que $\Delta(z)(i) = 0$ pour chaque $i \notin I$, on déduit de $\Delta(\sigma) + \vec{r} + \Delta(z) \geq \vec{0}$ que $\|\Delta(\sigma)\|^- \subseteq I$. Ce qui entraîne que $(\sigma_1, \ldots, \sigma_k, \sigma)$ est compensable. On considère l'ensemble I' des composantes augmentées par cette suite et on remarque que $\vec{p}', \vec{r}' \in \mathbb{Q}_{I'}^d$ et $z' \in (\mathbb{Q}_{I'}^d)^*$.

Remarquons que $u \in \Theta_I^* \vec{a}_1 \Theta_I^* \cdots \vec{a}_n \Theta_I^*$. Puisque $w = w'\sigma$, il existe $p \in \{1, \ldots, n\}$ tel que $w = \vec{a}_1 \cdots \vec{a}_{p-1}$ et $\sigma = \vec{a}_p \cdots \vec{a}_n$. On en déduit que u peut être décomposée en $u = u'\sigma'$ où $u' \in \Theta_I^* \vec{a}_1 \cdots \Theta_I^* \vec{a}_{p-1}$ et $\sigma' \in \Theta_I^* \vec{a}_p \Theta_I^* \cdots \vec{a}_n \Theta_I^*$. Observons que $\Delta(\sigma')(i) = \Delta(\sigma)(i)$ pour chaque $i \notin I$. Ainsi $\sigma' \in \Theta_{I'}^*$. On déduit de $\Theta_I^* \subseteq \Theta_{I'}^*$ que $u \in \Theta_{I'}^* \vec{a}_1 \Theta_{I'}^* \cdots \vec{a}_{p-1} \Theta_{I'}^*$.

Le lemme 5.35 montre qu'il existe une suite $n_1, \ldots, n_k \in \mathbb{N}$ telle que $\vec{v}_j = \Delta(\sigma_1^{n_1} \cdots \sigma_j^{n_j})$ est un vecteur de $\mathbb{N}^d$ et telle que $\|\vec{v}_k\|^+ = I$. Puisque $(\sigma_1, \ldots, \sigma_k)$ est insérable dans u, il existe une décomposition de u en $u = u_1 \cdots u_{k+1}$ et une suite $m_1, \ldots, m_k \in \mathbb{N}_{>0}$ telle que

$u_1\sigma_1^{m_1}\cdots u_k\sigma_k^{m_k}u_{k+1}$ est une trace depuis $\vec{c}_{\text{Init}}$. On observe par monotonie que pour tout $r \in \mathbb{N}$ il existe une configuration $\vec{c}_r$ telle que :

$$\vec{c}_{\text{Init}} \xrightarrow{u_1\sigma_1^{m_1+rn_1}\cdots u_k\sigma_k^{m_k+rn_k}u_{k+1}} \vec{c}_r$$

Considérons la configuration $\vec{c}'$ telle que $\vec{c}_{\text{Init}} \xrightarrow{u'} \vec{c}'$.

Montrons que $\vec{c}_r(i) \geq \vec{c}'(i)$ pour chaque $i \notin I$. On prend $i \notin I$. On introduit la configuration $\vec{c}$ telle que $\vec{c}' \xrightarrow{\sigma'} \vec{c}$. Remarquons que $\vec{c}_r(i) = \vec{c}(i)$ puisque $u_j \in \Theta_I^*$. On déduit de $\vec{c} = \vec{c}' + \Delta(\sigma')$ et de $\Delta(\sigma')(i) = \Delta(\sigma)(i)$, que $\vec{c}(i) = \vec{c}'(i) + \Delta(\sigma)(i)$. Comme $\|\Delta(\sigma)\|^- \subseteq I$ on obtient que $\vec{c}(i) \geq \vec{c}'(i)$. Ainsi $\vec{c}_r(i) \geq \vec{c}'(i)$ pour chaque $i \notin I$.

Considérons un entier $r \in \mathbb{N}$ suffisamment grand tel que $r \geq \vec{c}'(i)$ pour tout $i \in I$. Comme $\vec{c}_r(i) \geq r$ pour tout $i \in I$ on obtient $\vec{c}_r \geq \vec{c}'$. Puisque σ' est une trace depuis $\vec{c}'$ on en déduit que σ' est aussi une trace depuis $\vec{c}_r$. Le lemme 5.36 montre qu'il existe $r' \in \mathbb{N}$ tel que σ est une trace depuis $\vec{c}_r + r'\vec{v}_k$. On en déduit que $u_0\sigma_1^{m_1+r+r'}u_1\cdots\sigma_k^{m_k+r+r'}u_k\sigma$ est une trace depuis $\vec{c}_{\text{Init}}$. Ainsi $(\sigma_1, \ldots, \sigma_k, \sigma)$ est insérable dans u. □

Le lemme suivant montre qu'un témoin «partiel» de non-algébricité peut être obtenu à partir de toute exécution de $\mathcal{P}_W$. Ce témoin «partiel» est inductif et procurera un témoin de non-algébricité lorsque la configuration cible peut exécuter une transition d'«échec par pile vide». La preuve s'obtient par une induction immédiate sur la taille de l'exécution. La relation bien parenthésée R introduite dans ce lemme correspond intuitivement à la correspondance entre les opérations d'empilement et de dépilement le long de l'exécution. Le contenu de la pile est obtenu à partir des éléments correspondant aux *indices libres* de R. Un indice $j \in \{1, \ldots, k\}$ est dit *libre* pour une relation bien parenthésée R sur $\{1, \ldots, k\}$ si il n'existe pas de couple $(s, t) \in R$ satisfaisant $s < j < t$.

Lemme 5.38. *Pour toute configuration $(\vec{p}, w, \vec{r}, z)$ accessible dans $\mathcal{P}_W$ pour une exécution donnée, il existe un schéma R pour la suite $(\sigma_1, \ldots, \sigma_k)$ de circuits extraite le long de l'exécution et une suite $j_1 < \cdots < j_m$ d'indices libres pour R tels que :*

- *R satisfait la condition (5.5.3) de la définition 5.11,*
- *$\|\vec{r}\|^+ = \|\text{exc}(R)\|^+$,*
- *L'ensemble des mots $(\mathbb{Q}_{>0}\Delta(\sigma_{j_1}))\cdots(\mathbb{Q}_{>0}\Delta(\sigma_{j_m}))$ contient z si $\vec{p} = \vec{0}$ et il contient $z\vec{p}$ dans le cas contraire, et*
- *$j_m = k$ si $\vec{p} \not\geq \vec{0}$.*

Lorsque l'état $\lightning$ est accessible dans $\mathcal{P}_W$, un témoin de non-algébricité peut être déduit du témoin «partiel» introduit dans le lemme précédent.

Proposition 5.39. *Si l'état $\lightning$ est accessible dans $\mathcal{P}_W$ alors $\langle\Theta, \vec{c}_{\text{Init}}\rangle$ admet un témoin de non-algébricité.*

Démonstration. Il existe une configuration $(\vec{p}, w, \vec{r}, z)$ accessible dans $\mathcal{P}_W$ par une exécution étiquetée par u telle que $\vec{p} \not\geq \vec{0}$ et $\Delta(z) = \vec{0}$. Puisque le vecteur $\vec{p}+\vec{z}+\Delta(z)$ est constant lors d'un «traitement de vecteur non-positif» et il est dans $\mathbb{Q}^d_{\geq 0}$ juste après une transition d'«extraction de circuit», on déduit que $\vec{p} + \vec{r} + \Delta(z) \geq \vec{0}$. On déduit donc de $\Delta(z) = \vec{0}$ que $\|\vec{p}\|^- \subseteq \|\vec{r}\|^+$. Le lemme 5.37 montre que la suite $(\sigma_1, \ldots, \sigma_k)$ de circuits extraite le long de l'exécution est insérable dans U. Le lemme 5.38 montre qu'il existe un schéma R pour $(\sigma_1, \ldots, \sigma_k)$ satisfaisant la condition (5.5.3) de la définition 5.11 et tel que $\|\vec{r}\|^+ = \|\text{exc}(R)\|^+$ et $\vec{p} \in \mathbb{Q}_{>0}\Delta(\sigma_k)$.

Il existe une décomposition de u en $u = u_1\cdots u_{k+1}$ et une suite $m_1, \ldots, m_k \in \mathbb{N}_{>0}$ telles que $u_1\sigma_1^{m_1}\cdots u_k\sigma_k^{m_k}u_{k+1}$ est une trace depuis $\vec{c}_{\text{Init}}$. On prend le tuple $(u_1, \sigma_1^{m_1}, \ldots, u_k, \sigma_k^{m_k}, U)$ où U est la relation de correspondance $U = \{(2s, 2t) \mid (s, t) \in R\}$. Il reste à remarquer que ce tuple est un témoin de non-algébricité pour $\langle\Theta, \vec{c}_{\text{Init}}\rangle$. □

D'après la proposition 5.27, le langage des traces d'un système d'addition de vecteurs qui admet un témoin de non-algébricité n'est pas algébrique. On en déduit le corollaire suivant :

Corollaire 5.40. *Si l'état $\lightning$ est accessible dans $\mathcal{P}_W$ alors le langage des traces de $\langle \Theta, \vec{c}_{\text{Init}} \rangle$ est non-algébrique.*

Décidabilité du problème d'algébricité d'un système d'addition de vecteurs On montre maintenant en comment on peut utiliser l'automate à pile de vecteurs $\mathcal{P}_W$ introduit dans le paragraphe précédent pour résoudre le problème d'algébricité pour un système d'addition de vecteurs. Il y a deux causes possible d'échec pour $\mathcal{P}_W$. Le corollaire 5.40 montre que le langage des traces de $\langle \Theta, \vec{c}_{\text{Init}} \rangle$ n'est pas algébrique quand $\lightning$ est accessible dans $\mathcal{P}_W$. Toutefois, l'accessibilité de $\bot$ dans $\mathcal{P}_W$ signifie seulement, intuitivement, que l'ensemble support W est trop petit. On montre qu'il existe toujours un support fini suffisamment grand.

Proposition 5.41. *Il existe un ensemble support fini $W \subseteq \Theta^*$ tel que $\bot$ n'est pas accessible dans $\mathcal{P}_W$.*

Démonstration. Remarquons que $\bot$ n'est pas accessible dans $\mathcal{P}_{\Theta^*}$. Soit W l'ensemble de tous les mots $w \in \Theta^*$ tels que l'état $(\vec{0}, w)$ est accessible dans $\mathcal{P}_{\Theta^*}$. Il est aisé de vérifier que W est non-vide et clos par préfixe. Observons que toute configuration $(\vec{0}, w, \vec{r}, z)$ accessible dans $\mathcal{P}_W$ est également accessible dans $\mathcal{P}_{\Theta^*}$. Ainsi, $\bot$ n'est pas accessible dans $\mathcal{P}_W$. Il reste à montrer que W est fini.

Supposons, par contradiction, que W est infini. Puisque Θ est fini, le lemme de König nous montre qu'il existe une suite infinie $\vec{a}_1, \vec{a}_2, \dots$ d'actions telles que $w_n = \vec{a}_1 \cdots \vec{a}_n \in W$ pour tout $n \in \mathbb{N}$. Soit $n \in \mathbb{N}$. Par définition de W, l'état $(\vec{0}, w_{n+1}) = (\vec{0}, w_n \vec{a}_{n+1})$ est accessible dans $\mathcal{P}_{\Theta^*}$. Il s'ensuit que $\mathcal{P}_{\Theta^*}$ contient une exécution de la forme

$$(\vec{0}, \varepsilon, \vec{0}, \varepsilon) \xrightarrow{*} (\vec{0}, w_n, \vec{r}_n, z_n) \xrightarrow{\mathsf{test}(\neg Extr(w_n))} \cdots \tag{5.5.11}$$

On définit $\vec{v}_n = \mathsf{val}(\vec{0}, w_n, \vec{r}_n, z_n)$. On déduit de (5.5.10) que $\vec{v}_n \in \mathbb{N}^d$ pour chaque $n \in \mathbb{N}$. Le lemme de Dickson montre qu'il existe des indices $m < n$ tels que $\vec{v}_m \leq \vec{v}_n$. Il s'ensuit que

$$\Delta(w_m) + \vec{r}_m + \Delta(z_m) \;\leq\; \Delta(w_n) + \vec{r}_n + \Delta(z_n)$$

En conséquence, $\vec{r}_n + \Delta(z_n) + \Delta(\sigma) \geq \vec{0}$, où $\sigma = \vec{a}_{m+1} \cdots \vec{a}_n$. Observons que σ est un suffixe non-vide de w_n. On obtient que $\vec{r}_n, \Delta(z_n) \models \mathit{Extr}(w_n)$, ce qui contredit (5.5.11). □

Même dans le cas où l'ensemble support W est fini, $\mathcal{P}_W$ possède une infinité d'états et de transitions, ce qui est inadéquat dans une perspective algorithmique. Pour résoudre ce problème, on réduit $\mathcal{P}_W$ à son ensemble accessible d'états et de transitions. On définit Q^r_W et T^r_W les ensembles d'états et de transitions accessibles dans $\mathcal{P}_W$. Il est clair que l'automate à pile de vecteur réduit $\mathcal{P}^r_W = \langle Q^r_W, (\vec{0}, \varepsilon), \Theta, T^r_W \rangle$ ainsi obtenu possède les mêmes exécutions à partir de la configuration initiale que $\mathcal{P}_W$. Ces deux automates à pile de vecteurs reconnaissent donc le même langage. Toutefois, $\mathcal{P}^r_W$ contient toujours des vecteurs de rationnels dans les états et dans la pile. On commence par établir une condition suffisante pour que les composantes de ces vecteurs soient entières.

Lemme 5.42. *Pour toute configuration $(\vec{p}, w, \vec{r}, \vec{\gamma}_1 \cdots \vec{\gamma}_h)$ accessible dans $\mathcal{P}_W$, les composantes $\vec{p}(i)$ et $\vec{\gamma}_1(i), \dots, \vec{\gamma}_h(i)$ sont entières pour tout indice i tel que $\vec{r}(i) = 0$.*

Démonstration. Le lemme est trivialement vrai pour la configuration initiale $(\vec{0}, \varepsilon, \vec{0}, \varepsilon)$. On montre que la condition du lemme est préservée par chaque pas de $\mathcal{P}_W$. Considérons un pas $(\vec{p}, w, \vec{r}, z) \xrightarrow{t} (\vec{p}', w', \vec{r}', z')$ où t est une transition de $\mathcal{P}_W$, et supposons que le lemme est vrai pour $(\vec{p}, w, \vec{r}, z)$. On effectue une analyse de cas sur la transition t.

Si t est une transition de «traitement de vecteur non-positif», il dépile un vecteur $\vec{\gamma} \in \mathbb{Q}^d_{\geq 0}$ de la pile z et ajoute $\mathsf{rem}(\vec{\gamma}, \vec{p})$ au compteur r. On obtient que $\vec{p}' = \mathrm{mat}(\vec{\gamma}, \vec{p})$, $\vec{r}' = \vec{r} + \mathsf{rem}(\vec{\gamma}, \vec{p})$, et que $z = z'\vec{\gamma}$. Rappelons que $\vec{0} \leq \vec{r}$ et que $\vec{0} \leq \mathsf{rem}(\vec{\gamma}, \vec{p})$. Considérons un indice $i \in \{1, \ldots, d\}$ tel que $\vec{r}'(i) = 0$. Le fait que $\vec{r}' = \vec{r} + \mathsf{rem}(\vec{\gamma}, \vec{p})$ nous permet de déduire que $\vec{r}(i) = \mathsf{rem}(\vec{\gamma}, \vec{p})(i) = 0$. D'après l'hypothèse d'induction, $\vec{p}(i)$ et $\vec{\gamma}(i)$ sont deux entiers. On rappelle que $\vec{\gamma} + \vec{p} = \mathrm{mat}(\vec{\gamma}, \vec{p}) + \mathsf{rem}(\vec{\gamma}, \vec{p})$. On en déduit que $\vec{p}'(i) = \vec{\gamma}(i) + \vec{p}(i)$ est un entier, et comme z' est un préfixe de z, que le lemme est vrai pour $(\vec{p}', w', \vec{r}', z')$.

Les autres cas pour t dérivent immédiatement de l'hypothèse d'induction. □

Proposition 5.43. *Pour tout ensemble support fini W, les ensembles Q^r_W et T^r_W sont finis et calculables à partir de W et $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$.*

Démonstration. On introduit deux ensembles, $\vec{\Gamma}_W \subseteq \mathbb{Q}^d_{\geq 0}$ et $\vec{M}_W \subseteq \mathbb{Q}^d$, et on montre qu'ils contiennent les ensembles de symboles de pile accessible et de vecteurs $\vec{p}$ accessibles. Premièrement, $\vec{\Gamma}_W$ est l'ensemble de tous les vecteurs $\frac{k}{\Delta(\sigma)(i)}\Delta(\sigma)$ tels que

- σ est un suffixe d'un mot de W tel que $\Delta(\sigma) \geq \vec{0}$,
- i est un indice de $\{1, \ldots, d\}$ avec $\Delta(\sigma)(i) > 0$, et
- $k \in \{1, \ldots, \Delta(\sigma)(i)\}$.

Deuxièmement, $\vec{M}_W$ est le plus petit sous-ensemble de $\mathbb{Q}^d$ satisfaisant les deux conditions suivantes :

- $\Delta(\sigma) \in \vec{M}_W$ pour tout $w \in W$ et tout suffixe σ de w,
- $\mathrm{mat}(\vec{\gamma}, \vec{m}) \in \vec{M}_W$ pour tout $\vec{\gamma} \in \vec{\Gamma}_W$ et $\vec{m} \in \vec{M}_W$ avec $\vec{m} \not\geq \vec{0}$.

Il est aisé de vérifier que $\vec{\Gamma}_W$ est fini et calculable. Il existe donc $\eta > 0$ tel que $\eta \leq \vec{\gamma}(i)$ pour tout $\vec{\gamma} \in \vec{\Gamma}_W$ et $i \in \|\vec{\gamma}\|^+$. Observons que pour tout $\vec{\gamma} \in \vec{\Gamma}_W$ et $\vec{m} \in \mathbb{Q}^d$ avec $\vec{m} \not\geq \vec{0}$, si $\vec{m} \neq \mathrm{mat}(\vec{\gamma}, \vec{m})$ et $\mathrm{mat}(\vec{\gamma}, \vec{m}) \not\geq \vec{0}$ alors $\vec{m}(i) \leq \mathrm{mat}(\vec{\gamma}, \vec{m})(i)$ pour tout $i \in \|\vec{m}\|^-$ et $\vec{m}(j) + \eta \leq \mathrm{mat}(\vec{\gamma}, \vec{m})(j)$ pour un $j \in \|\vec{m}\|^-$. Il s'ensuit que $\vec{M}_W$ est également fini, et donc calculable.

Montrons que $\vec{p} \in \vec{M}_W$ et $z \in \vec{\Gamma}^*_W$ pour chaque configuration $(\vec{p}, w, \vec{r}, z)$ qui est accessible dans $\mathcal{P}_W$. La preuve est réalisée par induction sur la longueur des exécutions de $\mathcal{P}_W$ à partir de sa configuration initiale. Celle-ci, $(\vec{0}, \varepsilon, \vec{0}, \varepsilon)$, satisfait trivialement la propriété désirée. On considère une exécution

$$(\vec{0}, \varepsilon, \vec{0}, \varepsilon) \xrightarrow{*} (\vec{p}, w, \vec{r}, z) \xrightarrow{t} (\vec{p}', w', \vec{r}', z')$$

où t est une transition de $\mathcal{P}_W$, et on suppose que $\vec{p} \in \vec{M}_W$ et que $z \in \vec{\Gamma}^*_W$. Une inspection des règles de transitions de $\mathcal{P}_W$ montre que $\vec{p}' \in \vec{M}_W$ et $z' \in \vec{\Gamma}^*_W$. On détaille uniquement le cas non-trivial.

Si t est une transition «traitement d'un vecteur non-positif», alors $\vec{p} \not\geq \vec{0}$ et le vecteur $\vec{\gamma}$ qui est retiré de la pile satisfait $z = z'\vec{\gamma}$. Puisque $z \in \vec{\Gamma}^*_W$, on a que $\vec{\gamma} \in \vec{\Gamma}_W$. La définition de $\vec{M}_W$ implique directement que $\vec{p}' = \mathrm{mat}(\vec{\gamma}, \vec{p}) \in \vec{M}_W$.

Si t est une transition de «traitement de vecteur positif» et que $\|\vec{p}\|^+ \not\subseteq \|\vec{r}\|^+$, alors $\vec{r}(i) = 0 < \vec{p}(i)$ pour un indice $i \in \{1, \ldots, d\}$. Le lemme 5.42 montre que $\vec{p}(i) \in \mathbb{N}$. De plus, comme $\vec{p} \in \vec{M}_W$ et $\vec{p} > \vec{0}$, on obtient que $\vec{p} = \mu\Delta(\sigma)$ où σ est un suffixe d'un mot de W qui vérifie $\Delta(\sigma) \geq \vec{0}$, et μ est un rationnel tel que $0 < \mu \leq 1$. Remarquons que $\Delta(\sigma)(i) > 0$ et $\mu = \frac{\vec{p}(i)}{\Delta(\sigma)(i)}$. Il s'ensuit que $\vec{p} \in \vec{\Gamma}_W$, et ainsi que, $z' = z\vec{p}$ est contenu dans $\vec{\Gamma}^*_W$.

On a montré que $\vec{p} \in \vec{M}_W$ pour tout état $(\vec{p}, w)$ de Q^r_W, et $\vec{\gamma} \in \vec{\Gamma}_W$ pour chaque opération $\mathrm{push}_{\vec{\gamma}}$ ou $\mathrm{pop}_{\vec{\gamma}}$ de T^r_W. Ainsi, en restreignant $\vec{\gamma}$ et $\vec{p}$ à $\vec{\Gamma}_W$ et $\vec{M}_W$, respectivement, on obtient un ensemble $Q^\sharp_W$ d'états de $\mathcal{P}_W$ fini et calculable et un ensemble $T^\sharp_W$ de transitions de $\mathcal{P}_W$ fini et calculable tel que $Q^r_W \subseteq Q^\sharp_W$ et $T^r_W \subseteq T^\sharp_W$. Les ensembles Q^r_W et T^r_W sont donc tous les deux finis. Il est clair que l'automate à pile de vecteurs $\mathcal{P}^\sharp_W = \langle Q^\sharp_W, (\vec{0}, \varepsilon), \Theta, T^\sharp_W \rangle$ a les mêmes

$\mathsf{ContextFree}\,(\Theta, \vec{c}_{\mathrm{Init}})$

```
1   foreach finite support set W ⊆ Θ*
2 do
3         if ⊥ is not reachable in P_W
4 then
5               if ↯ is reachable in P_W
6 then
7                     return no
8               else
9                     return yes
```

FIGURE 5.10 – Algorithme résolvant le problème de l'algébricité d'un système d'addition de vecteurs.

exécutions depuis la configuration initiale que $\mathcal{P}_W$. La proposition 5.32 montre que les ensembles finis Q^r_W et T^r_W sont calculables. □

Nous avons maintenant tous les ingrédients nécessaires pour présenter notre algorithme résolvant le problème d'algébricité d'un système d'addition de vecteurs (cf figure 5.10). L'algorithme cherche un support fini W suffisamment grand, c.à.d tel que $\bot$ est inaccessible dans $\mathcal{P}_W$. L'algorithme renvoie ensuite `yes` si $\mathcal{P}_W$ réussit, et `no` s'il échoue. La proposition 5.43 garantit que les tests effectués aux lignes 4 et 6 sont calculables. La terminaison de cet algorithme est démontrée par la proposition 5.41. Sa correction provient du corollaire 5.40 pour la ligne 7, et des propositions 5.32 et 5.34 pour la ligne 9. Ceci conclut la preuve que le problème d'algébricité pour les systèmes d'addition de vecteur est décidable.

Remarque 5.2. *Quand* $\mathsf{ContextFree}\,(\Theta, \vec{c}_{\mathrm{Init}})$ *renvoie* `yes`*, un ensemble support fini* $W \subseteq \Theta^*$ *tel que* $\mathcal{P}_W$ *réussit a été calculé par l'algorithme. La proposition 5.43, montre que l'automate à pile de vecteurs réduit* $\mathcal{P}^r_W$ *a un nombre fini d'états et de transitions, et est calculable. On déduit de la proposition 5.32 que le langage des traces de* $\langle \Theta, \vec{c}_{\mathrm{Init}} \rangle$ *est effectivement algébrique.*

L'algorithme qu'on vient de présenter ne permet pas de dériver la complexité du problème d'algébricité d'un système d'addition de vecteurs. Dans [LPS13b], Leroux, Praveen et Sutre ont montré que ce problème est ExpSpace-complet.

Exemple 5.14. *On considère le système d'addition de vecteur* $\mathcal{V} = \langle \{\vec{a} = (2,-1), \vec{b} = (-1,2)\}, (1,1) \rangle$*. L'automate à pile de vecteur construit par la procédure décrite précédemment est donné à la figure 5.11. L'état* $\lightning$ *est accessible par le mot* $(\vec{a}\vec{b})\vec{a}\vec{b}$*. On a donc que* $(\vec{a}\vec{b}, \vec{a}, \vec{b}, \{(1,2)\})$ *est un témoin de non algébricité pour* $\mathcal{V}$*, comme vu dans l'exemple 5.13. En effet,* $\mathcal{L}(\mathcal{V}) \cap (\vec{a}\vec{b})^*\vec{a}^*\vec{b}^* = \{(\vec{a}\vec{b})^m \vec{a}^n \vec{b}^p \mid 0 \leq n \leq m+1 \wedge p \leq 2m+n+1\}$ *qui n'est pas algébrique.*

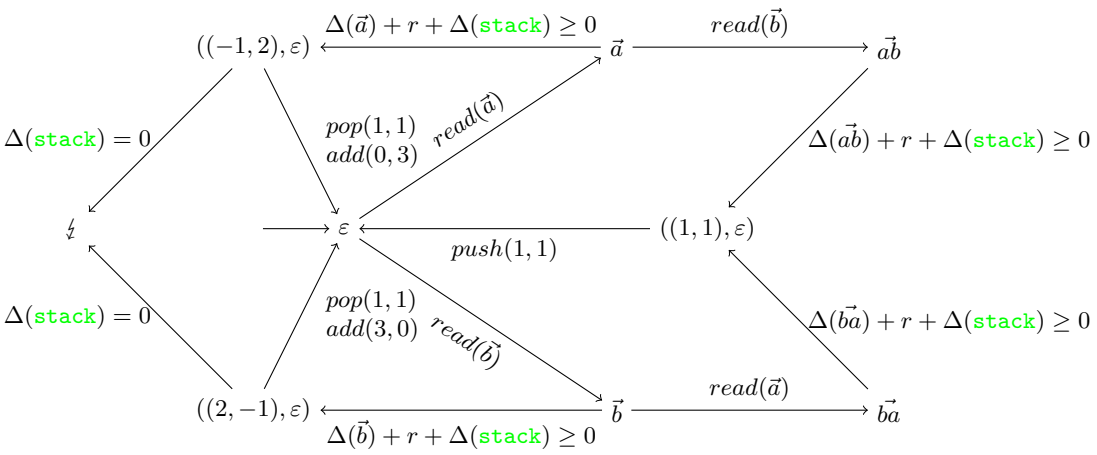


FIGURE 5.11 – Un automate à pile de vecteurs

Chapitre 6

Conclusion et perspectives

Nous résumons ici les résultats que nous avons obtenus dans le cadre de cette thèse, et nous présentons quelques pistes de recherches possibles à partir de ces résultats.

6.1 Autour des arbres de piles

Comme nous l'avions remarqué à la fin du chapitre 3, la figure 3.14 présentant les classes de graphes connus ne comportait aucun graphe située entre SR_n et TAut_n ayant une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable. Les résultats du chapitre 4 permettent notamment de compléter la figure 3.14 en la figure 6.1 : il existe bien à tout niveau des classes de graphes situées entre SR_n et TAut_n disposant d'une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable. Dans ce but, nous avons défini pour tout n l'ensemble ST_n des n-arbres de piles, que nous avons muni d'un ensemble d'opérations dans le but de définir une notion de réécriture suffixe. Nous avons démontré que ces systèmes de réécriture suffixe contiennent les systèmes de réécriture suffixe d'arbres et les systèmes de réécriture de pile. Nous avons ensuite présenté une notion de reconnaissabilité sur les opérations composées, à travers des automates d'opérations, étendant celle des opérations de piles et permettant la définition d'une notion de reconnaissabilité sur les n-arbres de piles. Nous en avons étudié les propriétés de clôture. De plus, nous avons montré que comme dans le cas des piles, il est possible de normaliser les automates d'opérations de manière à éviter les circuits dans les calculs. Nous avons ensuite défini formellement les trois classes de graphes associées à des systèmes de réécriture suffixe d'arbres de piles et aux transducteurs suffixes d'arbres de piles. Enfin, en se basant sur la notion de normalisation des automates, nous avons démontré que ces graphes disposent d'une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable, grâce à une interprétation à ensemble fini depuis un graphe suffixe-reconnaissable d'ordre n.

À partir de cette définition des arbres de piles et des systèmes de réécriture suffixe liés, plusieurs questions se posent.

Question 1 : Autres présentations

Une première piste de recherche serait de déterminer si les classes de graphes définies par les systèmes de réécriture suffixe d'arbres de piles disposent d'une autre représentation que la représentation interne que nous avons présentée. Une piste serait d'étudier l'extension des systèmes d'équations de graphes à des systèmes infinis d'équations, représentés par des graphes de la hiérarchie suffixe-reconnaissable. L'idée est que les systèmes finis d'équations sont représentés par des graphes finis étiquetés par les opérations VR, VRA ou VRS. Étant donné un graphe $\mathcal{G}$ étiqueté par des opérations VR, VRA ou VRS, le graphe $\mathcal{G}'$ qu'il définit est obtenu en considérant le terme (infini) obtenu en dépliant $\mathcal{G}$. Ainsi, $\mathcal{G}'$ est le plus petit point fixe satisfaisant ce

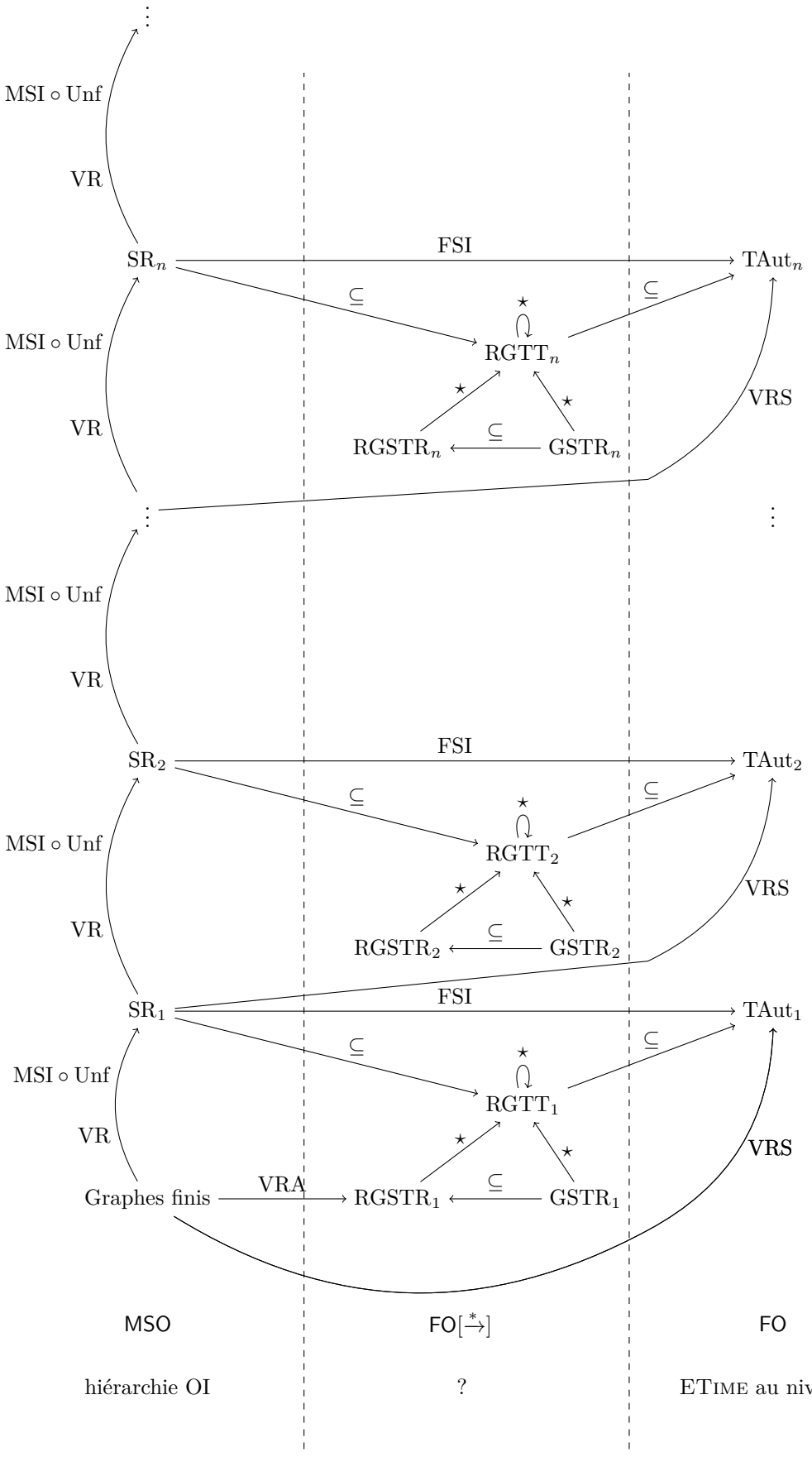


FIGURE 6.1 – Les hiérarchies de graphes étudiés dans ce document.

terme. Dans [CC03], Carayol et Colcombet ont montré qu'on peut considérer des graphes infinis étiquetés par des opérateurs VR (ils considèrent en réalité une version plus générale de ces opérateurs, qui est équivalente), et que si une classe de graphes $\mathcal{C}$ est close par MSO-interprétations, alors la classe de graphe définie par des graphes de $\mathcal{C}$ étiquetés par des opérateurs VR est égale à celle définie par les MSO-interprétation des dépliages des graphes de $\mathcal{C}$. Cela permet notamment d'affirmer que les graphes de SR_n sont les solutions des graphes de SR_{n-1} étiquetés par des opérateurs VR. Dans [Col04], Colcombet montre qu'on peut étendre cette technique aux VRS-systèmes d'équations, et obtenir ainsi les graphes de la hiérarchie arbre-automatique. Si on s'intéresse aux VRA-systèmes d'équations [Col02], on peut également considérer les graphes définis à partir de graphes de SR_n étiquetés par des opérateurs VRA. Par ailleurs, Colcombet montre directement qu'un tel graphe a une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable si le graphe étiqueté par des opérateurs VRA le définissant dispose d'une MSO-théorie décidable. Cela permet donc de définir une hiérarchie de graphes disposant d'une $\mathsf{FO}[\xrightarrow{*}]$-théorie décidable qui est définie à partir de la hiérarchie suffixe-reconnaissable. Nous pensons qu'étudier les liens entre ces graphes définis par des VRA-systèmes d'équations sur SR_n et les graphes définis par des systèmes de réécriture suffixe de piles est pertinent.

Nous conjecturons que $\mathrm{VRA}(\mathrm{SR}_n) = \mathrm{RGSTR}_{n+1}$. Pour obtenir ce résultat, la preuve fournie dans [Col02] n'est pas adaptable aux graphes de RGSTR_n, puisque la preuve construit des systèmes de réécriture suffixe d'arbres. Cependant, nous pensons que ce résultat peut être obtenu en comparant les interprétations à ensembles finis définissant les graphes depuis un graphe de SR_{n-1}.

Une autre piste consiste à s'intéresser aux graphes arbre-automatiques d'ordre supérieur. Au niveau 1, les graphes arbre-automatiques disposent d'une représentation interne : les arcs sont définis par des relations reconnues par des automates sur des superpositions d'arbres. Peut-on trouver un équivalent sur les arbres de piles d'ordre n, et la classe de graphes ainsi définie est-elle la classe des arbre-automatiques d'ordre n ? Cependant, pour pouvoir définir une telle notion, il est nécessaire de définir une notion de superposition des piles d'ordre supérieur et sur les arbres de piles (ces deux notions devraient être liées), et une telle notion n'est pas triviale : la difficulté provient du fait que deux piles d'ordre supérieur à 1 peuvent avoir des domaines ne différant pas que sur la fin. Prenons par exemple les 2-piles $p = [[\alpha\alpha\alpha]_1]_2$ et $p' = [[\beta]_1[\alpha]_1[\beta]_1]_2$: p possède une unique pile d'ordre 1, qui contient elle-même trois lettres quand p' possède 3 piles d'ordre 1, chacune contenant une unique lettre. Il apparaît qu'une superposition naïve demanderait d'avoir des trous pour les composantes représentant p' pour la 1-pile la plus basse, et des trous pour celles représentant p pour les autres. Une idée pour les superposer pourrait être d'avoir différents symboles pour représenter des trous de niveaux différents, mais une telle notion n'existe pas à notre connaissance.

Question 2 : Langages

Nous avons déjà cité des conjectures sur les classes de langages reconnus par les graphes définis par les systèmes de réécriture d'arbres de piles. Une piste de recherche pourrait être une étude plus systématique de ces classes de graphes, et aussi de ceux des graphes arbre-automatiques d'ordre supérieur. Suite à une discussion avec Thomas Colcombet, nous conjecturons que si les descriptions équationnelles de ces classes de graphes décrites à la fin du chapitre 4 se vérifient, nous pourrions en dériver une description des classes de langages reconnus. En effet, Colcombet a déterminé un lien entre les solutions d'un système d'équations et les langages reconnus par les solutions du système, et ces résultats ne dépendent a priori pas du caractère fini de ces systèmes d'équations. Toutefois, ce résultat n'a jamais été publié.

Question 3 : Arbres d'arbres

Une dernière piste serait une définition de la notion d'arbre d'arbres et de systèmes de réécriture suffixe associés. Une définition formelle d'un arbre d'arbres s'inspirant des arbres de

piles ne présente aucune difficulté :

- Un arbre d'ordre 0 est une lettre,
- Un arbre d'ordre $n+1$ est un arbre dont les nœuds sont étiquetés par des arbres d'ordre n.

On peut également aisément définir des opérations de base à tout niveau en s'inspirant des arbres de piles :

- Au niveau 0, on a les $\text{rew}_{\alpha,\beta}$,
- Au niveau n, on a les opérations copy_n^d et $\overline{\text{copy}}_n^d$.

Cependant, il sera moins aisé de définir une bonne notion d'opérations composées permettant à la fois d'avoir un certain contrôle sur l'application des opérations de base la composant et de ne pas être Turing-complet (un tel modèle assurant de fait l'indécidabilité de toute logique sur les graphes générés). Le seul moyen de définir de telles opérations composées serait de reprendre la définition des opérations composées sur les arbres de piles en remplaçant les opérations de base par celles sur les arbres d'arbres, et donc, toutes les opérations de niveau inférieur à n ne peuvent être localement composées entre elles, puisque contrairement au cas des piles, un arbre dispose d'un nombre arbitraire de sites où appliquer une opération. Autoriser la composition locale d'opérations de niveau inférieur à n permettrait de simuler un automate à deux compteurs, qui est un modèle Turing-complet. Ne pas l'autoriser mène à un modèle difficile à évaluer, étant donné que l'application des opérations devient dur à se représenter.

Par ailleurs, la preuve de décidabilité de $\mathsf{FO}[\xrightarrow{*}]$ ne s'étend pas de manière immédiate aux arbres d'arbres. En effet, il faudrait trouver pour cela un codage des arbres d'arbres par un ensemble de piles de piles tel que l'effet des opérations d'arbres d'arbres sur ces codages soit définissable en MSO. Si il est possible de trouver des codages qui à l'instar de celui des arbres de piles contiennent tous les chemins d'un arbre d'arbres (bien que les codages que nous ayons explorés ne soient pas très naturels), nous n'avons pas trouvé de manière de décrire en MSO l'effet des opérations d'arbres d'arbres sur ces codages.

6.2 Autour des systèmes à compteurs

Le chapitre 5 était consacré à l'étude de l'algébricité des langages de systèmes à compteurs plats et des systèmes d'addition de vecteurs, et du problème lié de la stratifiabilité d'un ensemble semi-linéaire. Dans un premier temps, nous avons montré que si on se restreint au cas des polyèdres d'entiers, le problème de stratifiabilité est décidable, et nous en avons donné un processus de décision effectif. Par la suite, nous avons utilisé ce résultat pour montrer que l'algébricité du langage d'un système à compteurs plat est décidable, en montrant que l'image de Parikh d'un tel langage est toujours un polyèdre d'entiers, et en utilisant le résultat de Ginsburg disant qu'un langage borné est algébrique si et seulement si son image de Parikh est stratifiable. Enfin, nous avons montré que l'algébricité du langage d'un système d'addition de vecteurs est décidable, et nous en avons fourni un algorithme effectif.

Question 1 : Problème de stratifiabilité

La stratifiabilité d'un polyèdre d'entiers est un cas particulier de la stratifiabilité d'un ensemble semi-linéaire. À notre connaissance ce problème est toujours ouvert dans le cas général. Une piste de recherche serait de tenter d'éclairer ce problème à travers les techniques que nous avons développées pour résoudre le cas des polyèdres d'entiers. Alternativement, il peut être souhaitable de regarder si il existe d'autres sous-cas intéressants de ce problème où la stratifiabilité serait décidable.

Question 2 : Langages des systèmes à compteurs

Comme nous l'avons rappelé au chapitre 3, les langages algébriques forment le niveau 1 de la hiérarchie OI. On pourrait étudier s'il est décidable de déterminer si le langage d'un système d'addition de vecteurs ou d'un système à compteurs plat appartient à la hiérarchie OI. Dans cette optique, on pourrait également étudier les liens entre les langages de ces modèles et ceux des systèmes de réécriture suffixe d'arbres de piles (bien qu'il soit sans doute nécessaire de mieux comprendre ces langages au préalable).

Enfin, peut-on trouver une manière de caractériser les langages reconnus par des systèmes d'addition de vecteurs et par les systèmes à compteurs plats ?

Question 3 : Graphes des systèmes à compteurs

Enfin, une dernière piste serait d'étudier directement les propriétés structurelles des graphes de systèmes d'addition de vecteurs et de systèmes à compteurs plats. Comme nous l'avons rappelé, le problème d'accessibilité est décidable pour les systèmes d'addition de vecteurs, et donc également sur les graphes associés. Il existe déjà une étude sur la décidabilité de la FO-théorie (et plusieurs de ses variantes) sur plusieurs classes de graphes liés aux systèmes d'addition de vecteurs [DDMM11], où il est notamment prouvé que dans le cas de graphes de systèmes d'additions de vecteurs non restreints à l'ensemble d'accessibilité du système, FO est décidable. Ce résultat devient faux si on restreint le graphe à l'ensemble d'accessibilité.

On peut se demander si il est possible d'étudier ces propriétés sur les graphes de systèmes à compteurs plats.

La question suivante nous semble également intéressante pour éclairer ces classes de graphes : est-il possible de trouver d'autres représentations de ces classes de graphes ?

Enfin, un autre problème lié à notre étude sur les langages des systèmes d'addition de vecteurs et de systèmes à compteurs plats est de déterminer si un graphe de système d'additions de vecteurs est un graphe suffixe-reconnaissable :

Entrée : Un système d'addition de vecteurs $\mathcal{S}$.

Sortie : Est-ce que le graphe du système d'addition de vecteurs est un graphe suffixe-reconnaissable ?

Il est important de noter que ce problème n'est pas équivalent à l'algébricité d'un système d'addition de vecteurs. Il est en effet possible d'avoir un graphe n'étant pas un graphe suffixe-reconnaissable dont l'ensemble des traces est pourtant un langage algébrique. Par ailleurs, il semble que l'ensemble des systèmes à compteurs dont le graphe est un graphe suffixe-reconnaissable soit très restreint. L'exemple suivant présente un système d'addition de vecteurs plat dont le langage est algébrique alors que son graphe n'est pas suffixe-reconnaissable.

Exemple 6.1. *On choisit* $\Theta = \{\vec{a}_1 = (1,0), \vec{a}_2 = (0,1), \vec{a}_3 = (-1,0)\}$ *et* $\vec{c}_{\text{Init}} = (0,0)$*. Et on considère le système d'addition de vecteurs* $\langle \Theta, \vec{c}_{\text{Init}} \rangle$ *dont on intersecte le langage avec* $\vec{a}_1^* \vec{a}_2^* \vec{a}_3^*$*. Le graphe de ce système d'addition de vecteurs est le graphe* $\mathcal{G} = (V, E)$ *avec :*

$$\begin{aligned} V &= \mathbb{N} \times \mathbb{N} \times \{0,1,2\} \\ E &= \{((n,0,0), \vec{a}_1, (n+1,0,0)), ((n,0,0), \vec{a}_2, (n,1,1)), ((n,m,1), \vec{a}_2, (n,m+1,1)), \\ &\quad ((n,m,1), \vec{a}_3, (n-1,m,2)), ((n+1,m,2), \vec{a}_3, (n,m,2)), ((n+1,0,0), \vec{a}_3, (n,0,2)) \\ &\quad \mid n \geq 0, m \geq 0\} \end{aligned}$$

Ce graphe, représenté à la figure 6.2, n'est pas suffixe-reconnaissable. Cependant, le langage reconnu est $\{\vec{a}_1^n \vec{a}_2^m \vec{a}_3^p \mid p \leq n\}$*, qui est bel et bien algébrique.*

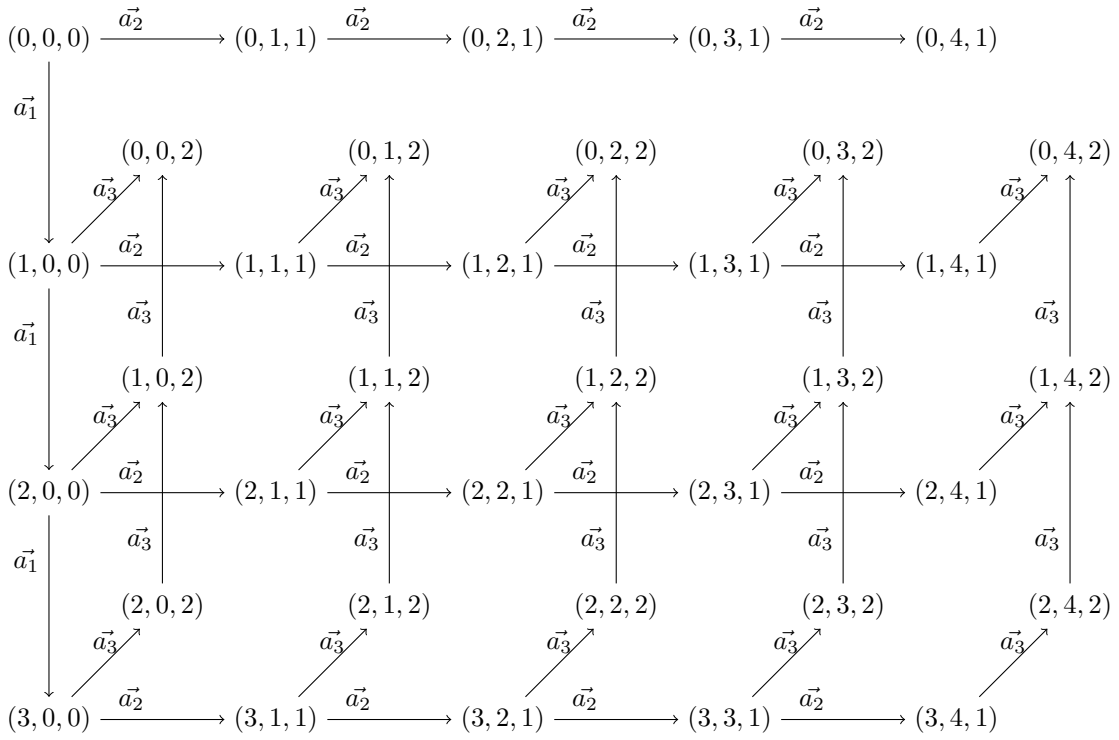


FIGURE 6.2 – Le graphe du système d'addition de vecteurs de l'exemple 6.1.

Bibliographie

[ABS01] A. Annichini, A. Bouajjani, and M. Sighireanu. TReX : A Tool for Reachability Analysis of Complex Systems. In *Proc. CAV'01*, volume 2102 of *LNCS*, pages 368–372. Springer, 2001.

[AG68] M. Arbib and Y. Give'on. Algebra Automata I : Parallel Programming as a Prolegomena to the Categorical Approach. *Information and Control*, 12(4) :331–345, April 1968.

[Aho68] A. Aho. Indexed Grammars - An Extension of Context-Free Grammars. *J. ACM*, 15(4) :647–671, 1968.

[Bar97] K. Barthelmann. On Equational Simple Graphs. Technical report, 1997.

[BFLP08] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST : acceleration from theory to practice. volume 10, pages 401–424. Springer, 2008.

[BG00] A. Blumensath and E. Grädel. Automatic Structures. In *LICS*, pages 51–62. IEEE Computer Society, 2000.

[BH99] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theoretical Computer Science*, 221(1–2) :211–250, 1999.

[BIK10] M. Bozga, R. Iosif, and F. Konecný. Fast Acceleration of Ultimately Periodic Relations. In *Proc. CAV'10*, volume 6174 of *LNCS*, pages 227–242. Springer, 2010.

[BL69] J. Büchi and L. Landweber. Definability in the Monadic Second-Order Theory of Successor. *J. Symb. Log.*, 34(2) :166–170, 1969.

[Blu99] A. Blumensath. *Automatic Structures.* PhD thesis, RWTH Aachen, 1999.

[Blu01] A. Blumensath. Prefix-Recognisable Graphs and Monadic Second-Order Logic, 2001.

[Bra68] W. Brainerd. The Minimalization of Tree Automata. *Information and Control*, 13(5) :484–491, November 1968.

[Bra69] W. Brainerd. Tree Generating Regular Systems. *Information and Control*, 14(2) :217–231, February 1969.

[BW94] B. Boigelot and P. Wolper. Symbolic Verification with Periodic Sets. In *Proc. CAV'94*, volume 818 of *LNCS*, pages 55–67. Springer, 1994.

[Car05] A. Carayol. Regular Sets of Higher-Order Pushdown Stacks. In *MFCS*, volume 3618 of *LNCS*, pages 168–179. Springer, 2005.

[Car06] A. Carayol. *Automates infinis, logiques et langages.* PhD thesis, Université de Rennes 1, 2006.

[Cau92] D. Caucal. On the Regular Structure of Prefix Rewriting. *Theor. Comput. Sci.*, 106(1) :61–86, 1992.

[Cau95] D. Caucal. *Bisimulation of context-free grammars and of pushdown automata*, volume 53, pages 85–106. 1995.

[Cau96] D. Caucal. On Infinite Transition Graphs Having a Decidable Monadic Theory. In *ICALP*, volume 1099 of *LNCS*, pages 194–205. Springer, 1996.

[Cau02] D. Caucal. On Infinite Terms Having a Decidable Monadic Theory. In Diks and Rytter [DR02], pages 165–176.

[CC03] A. Carayol and T. Colcombet. On Equivalent Representations of Infinite Structures. In J. Baeten, J. Lenstra, J. Parrow, and G. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 599–610. Springer, 2003.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on : `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *Proc. CAV'98*, volume 1427 of *LNCS*, pages 268–279. Springer, 1998.

[CK01] D. Caucal and T. Knapik. An Internal Presentation of Regular Graphs by Prefix-Recognizable Graphs. *Theory Comput. Syst.*, 34(4) :299–336, 2001.

[CK02] D. Caucal and T. Knapik. A Chomsky-Like Hierarchy of Infinite Graphs. In Diks and Rytter [DR02], pages 177–187.

[CL07] T. Colcombet and C. Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science (LMCS)*, 3(2), 2007.

[CLM76] E. Cardoza, R. Lipton, and A. Meyer. Exponential Space Complete Problems for Petri Nets and Commutative Semigroups : Preliminary Report. In A. Chandra, D. Wotschke, E. Friedman, and M. Harrison, editors, *STOC*, pages 50–54. ACM, 1976.

[Col02] T. Colcombet. On Families of Graphs Having a Decidable First Order Theory with Reachability. In P. Widmayer, F. Triguero Ruiz, R. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2002.

[Col04] T. Colcombet. *Représentation et propriétés de structures infinies*. PhD thesis, Université de Rennes 1, 2004.

[Cou89] Bruno Courcelle. The Monadic Second-Order Logic of Graphs, II : Infinite Graphs of Bounded Width. *Mathematical Systems Theory*, 21(4) :187–221, 1989.

[Cou90] B. Courcelle. Graph Rewriting : an Algebraic and Logic Approach. volume B of *Handbook of Theoretical Computer Science*, pages 193–242. Elsevier Science, 1990.

[CW98] B. Courcelle and I. Walukiewicz. Monadic Second-Order Logic, Graph Coverings and Unfoldings of Transition Systems. *Ann. Pure Appl. Logic*, 92(1) :35–62, 1998.

[CW03] A. Carayol and S. Wöhrle. The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In *FSTTCS*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.

[Dam82] W. Damm. The IO- and OI-Hierarchies. *Theor. Comput. Sci.*, 20 :95–207, 1982.

[DDMM11] P. Darondeau, S. Demri, R. Meyer, and C. Morvan. Petri Net Reachability Graphs : Decidability Status of FO Properties. In S. Chakraborty and A. Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 140–151. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[DHLT90] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. Decidability of the Confluence of Finite Ground Term Rewrite Systems and of Other Related Term Rewrite Systems. *Inf. Comput.*, 88(2) :187–201, 1990.

[Don65] J. Doner. Decidability of the weak second-order theory of two successors. *Notices American Mathematical Society*, 12 :365–468, March 1965.

[Don70] J. Doner. Tree Acceptors and Some of Their Applications. *J. Comput. Syst. Sci.*, 4(5) :406–451, 1970.

[DR02] K. Diks and W. Rytter, editors. *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *LNCS*. Springer, 2002.

[DT90] M. Dauchet and S. Tison. The Theory of Ground Rewrite Systems is Decidable. In *LICS*, pages 242–248. IEEE Computer Society, 1990.

[DTHL87] M. Dauchet, S. Tison, T. Heuillard, and P. Lescanne. Decidability of the Confluence of Ground Term Rewriting Systems. In *LICS*, pages 353–359. IEEE Computer Society, 1987.

[EGM12] J. Esparza, P. Ganty, and R. Majumdar. A Perfect Model for Bounded Verification. In *Proc. LICS'12*, pages 285–294. IEEE Computer Society, 2012.

[Eng91] J. Engelfriet. Iterated Stack Automata and Complexity Classes. *Inf. Comput.*, 95(1) :21–75, November 1991.

[EW67] S. Eilenberg and J. Wright. Automata in General Algebras. *Information and Control*, 11(4) :452–470, October 1967.

[FIS03] A. Finkel, S. Iyer, and G. Sutre. Well-Abstracted Transition Systems : Application to FIFO automata. *Information and Computation*, 181(1) :1–31, 2003.

[Fra05] S. Fratani. *Automates à piles de piles ... de piles*. PhD thesis, Université Bordeaux 1, 2005.

[Gin66] S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, 1966.

[Gre70] S. Greibach. Full AFLs and Nested Iterated Substitution. *Information and Control*, 16(1) :7–35, March 1970.

[GS64] S. Ginsburg and E. Spanier. Bounded ALGOL-like languages. *Trans. Amer. Math. Soc.*, 113 :333–368, 1964.

[GS66] S. Ginsburg and E. Spanier. Semigroups, Presburger formulas and languages. *Pacific J. Math.*, 16(2) :285–296, 1966.

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akadéniai Kiadó, Budapest, Hungary, 1984.

[GY80] A. Ginzburg and M. Yoeli. Vector Addition Systems and Regular Languages. *J. Comput. Syst. Sci.*, 20(3) :277–284, 1980.

[Hod83] B. Hodgson. Décidabilité par automate fini. *Ann. Sci. Math. Québec*, 7(1) :39–57, 1983.

[HP79] J. Hopcroft and J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Comput. Sci.*, 8(2) :135–159, 1979.

[KM69] R. Karp and R. Miller. Parallel Program Schemata. *J. Comput. Syst. Sci.*, 3(2) :147–195, 1969.

[KN95] B. Khoussainov and A. Nerode. Automatic presentations of structures. In D. Leivant, editor, *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer Berlin Heidelberg, 1995.

[KNU02] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-Order Pushdown Trees Are Easy. In *FoSSaCS*, volume 2303, pages 205–222. Springer, 2002.

[Kos82] R. Kosaraju. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In H. Lewis, B. Simons, W. Burkhard, and L. Landweber, editors, *STOC*, pages 267–281. ACM, 1982.

[Lam92] J. Lambert. A Structure to Decide Reachability in Petri Nets. *Theor. Comput. Sci.*, 99(1) :79–104, 1992.

[Ler10] J. Leroux. The General Vector Addition System Reachability Problem by Presburger Inductive Invariants. *Logical Methods in Computer Science*, 6(3), 2010.

[Ler13a] J. Leroux. Acceleration For Presburger Petri Nets. In A. Lisitsa and A. Nemytykh, editors, *VPT@CAV*, volume 16 of *EPiC Series*, pages 10–12. EasyChair, 2013.

[Ler13b] J. Leroux. Presburger Vector Addition Systems. In *LICS*, pages 23–32. IEEE Computer Society, 2013.

[Ler13c] J. Leroux. Vector Addition System Reversible Reachability Problem. *Logical Methods in Computer Science*, 9(1), 2013.

[Löd02] C. Löding. Model-Checking Infinite Systems Generated by Ground Tree Rewriting. In M. Nielsen and U. Engberg, editors, *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 280–294. Springer, 2002.

[Löd03] C. Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003.

[LPS13a] J. Leroux, V. Penelle, and G. Sutre. On the Context-Freeness Problem for Vector Addition Systems. In *Proc. LICS'13*, pages 43–52. IEEE Computer Society, 2013. http ://hal.archives-ouvertes.fr/hal-00788744.

[LPS13b] J. Leroux, M. Praveen, and G. Sutre. A Relational Trace Logic for Vector Addition Systems with Application to Context-Freeness. In *Proc. CONCUR'13*, volume 8052 of *LNCS*, pages 137–151. Springer, 2013.

[LPS14] J. Leroux, V. Penelle, and G. Sutre. The Context-Freeness Problem Is coNP-Complete for Flat Counter Systems. In F. Cassez and J. Raskin, editors, *ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2014.

[Mas74] A. Maslov. The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.*, 15 :1170–1174, 1974.

[Mas76] A. Maslov. Multi-level stack automata. *Problems Inform. Transmission*, 126 :38–43, 1976.

[May81] E. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *STOC*, pages 238–246. ACM, 1981.

[Mey08] A. Meyer. Traces of term-automatic graphs. *ITA*, 42(3) :615–630, 2008.

[Min67] M. Minsky. *Computation : Finite and Infinite Machines*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, 1967.

[MS85] D. Muller and P. Schupp. The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theor. Comput. Sci.*, 37 :51–75, 1985.

[MW67] J. Mezei and Jesse B. Wright. Algebraic Automata and Context-Free Sets. *Information and Control*, 11(1/2) :3–29, July-August 1967.

[Par61] R. Parikh. Language-Generating Devices. Technical report, Research Laboratory of Electronics, Massachusetts Institute of Technology, 1961.

[Par66] R. Parikh. On Context-Free Languages. *J. ACM*, 13(4) :570–581, 1966.

[Pen15] V. Penelle. Rewriting Higher-Order Stack Trees. In L. Beklemishev and D. Musatov, editors, *CSR*, volume 9139 of *Lecture Notes in Computer Science, Best Student Paper Award*, pages 364–397. Springer, 2015.

[Pet62] C. Petri. Fundamentals of a Theory of Asynchronous Information Flow. In *Information Processing 62*, pages 386–391. IFIP, North-Holland, 1962.

[Pre30] M. Presburger. über die Vollstandigkeit eines gewissen Systems der Arithmetic ganzer Zahlen, welchem die Addition als einzige Operation hervortritt, 1930.

[Rab69] M. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141 :1–35, July 1969.

[Rac78] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Comput. Sci.*, 6(2) :223–231, 1978.

[Sch87] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1987.

[Sch92] S. Schwer. The Context-Freeness of the Languages Associated with Vector Addition Systems is Decidable. *Theoretical Comput. Sci.*, 98(2) :199–247, 1992.

[Sem84] A. Semenov. Decidability of Monadic Theories. In M. Chytil and V. Koubek, editors, *MFCS*, volume 176 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 1984.

[She75] S. Shelah. The monadic second order theory of order. *Annals of Mathematics*, 102 :379–419, 1975.

[ST77] G. Sacerdote and R. Tenney. The Decidability of the Reachability Problem for Vector Addition Systems (Preliminary Version). In J. Hopcroft, E. Friedman, and M. Harrison, editors, *STOC*, pages 61–76. ACM, 1977.

[Sti98] C. Stirling. The Joys of Bisimulation. In L. Brim, J. Gruska, and J. Zlatuska, editors, *MFCS*, volume 1450 of *Lecture Notes in Computer Science*, pages 142–151. Springer, 1998.

[Sti00] C. Stirling. Decidability of bisimulation equivalence for pushdown processes. Technical Report EDI-INF-RR-0005, University of Edinburgh, 2000.

[Tha70] J. Thatcher. Generalized Sequential Machine Maps. *J. Comput. Syst. Sci.*, 4(4) :339–367, 1970.

[Tho90] W. Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B)*, pages 133–192. 1990.

[TW65] J. Thatcher and J. Wright. Generalized Finite Automata. *Notices American Mathematical Society*, 820, 1965.

[TW68] J. Thatcher and J. Wright. Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory*, 2(1) :57–81, 1968.

[VVN81] R. Valk and G. Vidal-Naquet. Petri nets and regular languages. *J. Comput. Syst. Sci.*, 23(3) :299–325, 1981.

[Wal96] I. Walukiewicz. Monadic Second Order Logic on Tree-Like Structures. In C. Puech and R. Reischuk, editors, *STACS*, volume 1046 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 1996.

[Wal02] I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.*, 275(1-2) :311–346, 2002.