Université de la Réunion

Institut de Recherche en Mathématiques et Informatique Appliquées

Produit Synchronisé pour Quelques Classes de Graphes Infinis

THÈSE

présentée et soutenue publiquement le 10 février 2000

pour l'obtention du

Doctorat de l'Université de la Réunion

Spécialité Informatique

par

Étienne PAYET

Composition du jury

Rapporteurs :	André ARNOLD Didier CAUCAL
Examinateurs :	Teodor KNAPIK Henri RALAMBONDRAINY Wolfgang THOMAS

Ecole Doctorale de Mathématiques et Informatique de Marseille

Remerciements

Cette thèse a été préparée à l'Institut de Recherche en Mathématiques et Informatique Appliquées *(IREMIA)* de l'Université de la Réunion. Je tiens à en remercier tous les membres pour leur accueil, tout particulièrement son Directeur, *Pierre Gigord*, pour son soutien et son dynamisme.

Teodor Knapik a encadré mes travaux avec une attention exemplaire durant mon année de DEA et mes trois années de thèse. Je le remercie chaleureusement pour sa grande disponibilité ainsi que les commentaires et suggestions pertinentes dont il m'a fait bénéficier.

C'est pour moi un grand honneur qu'*André Arnold* ait accepté de faire partie de mon jury en qualité de rapporteur. Ses remarques judicieuses lors de son évaluation de mon travail ont largement contribué à ameliorer ce document. D'autre part, le produit synchronisé de systèmes de transitions étiquetés qu'il a introduit avec Maurice Nivat est une opération fondamentale en informatique et a été la base de plusieurs parties de cette thèse.

Je remercie vivement *Didier Caucal* pour son soutien et son intérêt pour mes travaux. C'est pour moi une grande fierté qu'il ait accepté d'être rapporteur de ma thèse et de le compter parmi les membres de mon jury. Ses remarques profondes et pertinentes ont indiscutablement amélioré ce mémoire. Je souhaite lui déclarer mon admiration pour la qualité et l'importance de ses travaux qui ont tant contribué à la connaissance des graphes infinis.

La présence de *Wolfgang Thomas* dans mon jury de thèse est un honneur immense que j'ai peine à mesurer. Je souhaite lui témoigner ici ma profonde reconnaissance pour s'être déplacé si loin afin de juger mes travaux.

Je remercie sincèrement *Henri Ralambondrainy* d'avoir accepté d'être le Directeur cette thèse.

Je ne saurais trouver les justes mots pour remercier toute ma famille pour son soutien, son attention et ses encouragements durant mes longues années d'études. Ma gratitude et mon amour éternels à *Marie-Claire*, *Ludo*, *Léo* et *Pierre*. Ces modestes travaux vous sont dédiés.

Résumé

Cette thèse a pour cadre la spécification et la vérification de systèmes informatiques distribués, concurrents ou réactifs au moyen de graphes infinis associés à des spécifications de Thue et à certaines machines.

Nous montrons que la classe des graphes des spécifications de Thue est fermée par produit synchronisé. Nous établissons aussi ce fait pour la classe des graphes des machines de Turing et pour certaines de ses sous-classes.

Nous nous intéressons également à la conservation par produit synchronisé de la décidabilité de la théorie du premier ordre de graphes infinis. Nous montrons que le produit synchronisé de graphes de machines à pile restreint à sa partie accessible depuis un sommet donné a une théorie du premier ordre qui n'est pas décidable. Cependant, le produit synchronisé de graphes sans racine distinguée et dont la théorie du premier ordre est décidable a une théorie du premier ordre qui est décidable.

Enfin, nous mettons en évidence des liens qui unissent les graphes des machines de Turing à ceux des spécifications de Thue.

Mots-clés: Vérification Formelle, Graphes Infinis, Spécifications de Thue, Machines de Turing.

Abstract

The global framework of this thesis is specification and verification of distributed, concurrent or reactive systems by means of infinite graphs associated to Thue specifications and to some machines.

We prove that the class of graphs of Thue specifications is closed under synchronized product. This result is established as well for the class of graphs of Turing machines and for some of its subclasses.

We also investigate conservation of the decidability of the first-order theory of infinite graphs under synchronized product. We prove that the restriction of the product of graphs of pushdown machines to the part that is accessible from a given vertex has a non-decidable first-order theory. Nevertheless, the synchronized product of graphs with no distinguished root and the first-order theory of which is decidable has a decidable first-order theory.

Finally, we point out some links between graphs of Thue specifications and graphs of Turing machines.

Keywords: Formal Verification, Infinite Graphs, Thue Specifications, Turing Machines.

Table des matières

Introduction

Les systèmes informatiques jouent aujourd'hui un rôle majeur au sein de nombreux secteurs d'activité. Des enjeux économiques de productivité, de réactivité et d'efficacité reposent en effet entièrement sur une informatique de pointe dont l'importance est devenue cruciale au fil de ces dernières années. Par ailleurs, de nombreuses vies humaines dépendent entièrement de systèmes embarqués qui régissent par exemple le fonctionnement d'avions de ligne assurant quotidiennement le transport de milliers de personnes. Les plus petites erreurs lors de la phase de conception et de réalisation de logiciels tels que les protocoles de communication réseau, les systèmes d'exploitation, les systèmes de gestion de bases de données ou les programmes intervenant dans les systèmes embarqués peuvent entraîner des surcoûts considérables et provoquer des défaillances aux conséquences irréversibles, voire dramatiques.

Il n'est donc pas étonnant qu'un grand nombre de recherches se soient concentrées sur la mise au point de méthodes de vérification de systèmes distribués, concurrents ou encore réactifs qui sont au cœur du fonctionnement des secteurs d'activité de pointe. De tels systèmes jouent un rôle d'une importance croissante et sont complexes et difficiles à concevoir. Ils se distinguent nettement des programmes « classiques » car, dans le cas idéal, ils ne s'arrêtent jamais et maintiennent une interaction permanente avec leur environnement. Leur validation pose en général des problèmes pratiques et théoriques ardus qui soulèvent de véritables défis scientifiques.

La mise au point de techniques de tests intensifs a formé un premier pas vers la création de méthodes de vérification pratiques. L'efficacité de ces techniques reste toutefois limitée car elle repose sur la mise en place d'un ensemble de tests suffisamment large pour mettre en évidence la présence d'erreurs de conception. D'autre part, une fois les tests effectués, il faut interpréter la vaste quantité d'informations récoltées. Depuis quelques années, des techniques de vérification formelle sont venues compléter les méthodes de tests et ont permis de vérifier des systèmes d'une complexité impressionnante. Ces techniques sont utilisées en amont des méthodes de tests et permettent de détecter, dès la phase de conception, un certain nombre d'erreurs. L'avantage est donc qu'elles évitent les phases de correction coûteuses qui peuvent survenir après la conception et la réalisation de logiciels car elles interviennent très tôt dans le processus de développement. Un grand nombre de recherches leur sont aujourd'hui consacrées.

Vérification formelle

La *vérification formelle* consiste en la donnée d'un modèle mathématique d'un système, d'un langage permettant de spécifier les propriétés désirées du système et d'une méthode de preuve qui permet de vérifier que les propriétés spécifiées sont satisfaites.

Parmi les approches possibles de vérification formelle, figurent les méthodes de *preuve de théorèmes*, les méthodes de *model checking* et les méthodes d'*equivalence checking*. Les méthodes de preuve de théorèmes consistent à axiomatiser un modèle du système, à exprimer les propriétés désirées dans une logique convenable et à montrer, en construisant une preuve dans cette logique, que l'axiomatisation implique les propriétés. C'est une approche puissante et flexible mais qui devient très vite lourde et est au mieux semi-automatisable. Les méthodes de model checking consistent à représenter le système informatique sous la forme d'un système de transitions fini (c'est-à-dire un graphe dont les arcs sont étiquetés par les actions du système) et à vérifier que ce système de transitions est un modèle des propriétés qui sont exprimées par des formules d'une logique temporelle propositionnelle. L'avantage principal de cette approche est qu'elle est en général complètement automatisable. Elle est cependant limitée par la taille des systèmes de transitions considérés. Enfin, les méthodes d'equivalence checking reposent sur la notion d'équivalence observationnelle qui formalise le fait que deux processus présentent le même comportement. Ces méthodes cherchent à montrer qu'un système de processus et des spécifications donnés sont équivalents par rapport à une relation particulière.

L'étude présentée dans ce document ne concerne pas l'axiomatisation de structures mais leur représentation de façon finie et compacte sans utiliser de formules logiques. C'est pourquoi notre travail est plutôt rattaché aux méthodes de model checking bien que notre intérêt réside dans l'étude de graphes infinis. Edmund Clarke et Allen Emerson furent parmi les premiers à proposer un algorithme efficace de model checking pour une logique temporelle arborescente baptisée CTL [CE81]. Cet algorithme est basé sur la caractérisation des opérateurs de CTL au moyen de plus petits ou de plus grands points fixes de fonctions monotones. Il faut toutefois savoir que dès 1960, Richard Büchi avait publié des travaux relatifs au problème de décision pour une logique dont les formules sont interprétées sur des mots et dont la logique temporelle propositionnelle à temps linéaire constitue un fragment [Büc60].

L'algorithme de Clarke et Emerson nécessite de construire dans un premier temps un système de transitions complet représentant le système informatique considéré. L'inconvénient de cette approche est que lorsqu'on s'attaque à des cas pratiques réalistes, la taille des systèmes de transitions devient rapidement gigantesque et l'on a à faire face à ce qui est communément appelé l'*explosion du nombre des états*. Un certain nombre d'études ont été consacrées à la résolution de ce problème et ont donné naissance à diverses méthodes plus ou moins efficaces. Parmi ces méthodes, figurent les techniques de *réduction* du nombre des états d'un système de processus sans perdre d'information essentielle sur son fonctionnement. Robert Kurshan [Kur87] a dans cet objectif proposé toute une méthodologie permettant de construire une hiérarchie de réductions dans laquelle les processus au niveau le plus bas sont d'abord réduits puis sont combinés aux processus du niveau supérieur et sont à nouveau réduits, etc. L'inconvénient des méthodes de réduction est qu'elles sont en général difficiles à automatiser. Des études ont été menées dans le but de créer des techniques de réduction complètement automatisables, ce qui a donné naissance à des méthodes qui consistent à réduire un processus en un autre équivalent par rapport à des critères d'observabilité qui tiennent compte des entrées et des sorties du processus initial (voir [CLM89]). Cette réduction préserve la valeur de vérité de formules écrites dans une logique adéquate.

Le *model checking symbolique* est une autre approche qui a été développée dans le but de contourner le problème d'explosion du nombre des états. L'objectif est d'éviter de construire un système de transitions complet modélisant le système informatique avant d'effectuer des vérifications. Le système informatique est plutôt représenté implicitement au moyen d'une formule booléenne qui est manipulée efficacement grâce à des diagrammes de décision binaires [Bry86].

Le problème d'espace mémoire nécessaire à la représentation de systèmes informatiques pourrait également être surmonté grâce à l'utilisation de systèmes de transitions infinis. De tels systèmes présentant une grande régularité peuvent en effet être représentés de façon finie et compacte. Par conséquent, un processus dont le comportement est modélisé grâce à un système de transitions fini immense mais très régulier pourrait être manipulé efficacement par le biais d'une présentation compacte d'un graphe infini qui serait une approximation de son comportement. Une telle approche éviterait donc d'énumérer tous les états du processus dont le nombre peut atteindre des proportions considérables. Elle permettrait également de traiter efficacement le cas de processus dont le nombre d'états est réellement infini. Un certain nombre de recherches ont été menées dans le but d'identifier des classes de systèmes de transitions, ou de graphes, infinis sur lesquels il est possible d'effectuer de la vérification automatique. Les travaux exposés dans ce document s'inscrivent dans ce cadre et ont pour objectif l'étude de plusieurs classes de graphes infinis.

Systèmes de transitions infinis et produit synchronisé

Comme le fait remarquer André Arnold (par exemple dans [Arn92]), les systèmes de transitions étiquetées peuvent être considérés comme la notion fondamentale pour décrire formellement le fonctionnement d'un système de processus. Un *système de transitions étiquetées* se compose d'un ensemble d'états et d'un ensemble d'arcs étiquetés, appelés transitions, entre ces états. Ces transitions représentent les effets possibles de l'exécution d'une action à partir d'un état.

Le *produit synchronisé* de systèmes de transitions est une opération fondamentale introduite par André Arnold et Maurice Nivat [AN82, Niv79] qui permet de représenter un système de processus en tenant compte des interactions qui ont lieu dans le système. Étant donnés des systèmes de transitions $\mathcal{T}_1, \ldots, \mathcal{T}_n$ qui modélisent les processus, on s'intéresse au système de transitions dont l'ensemble des états est le produit des ensemble des états de $\mathcal{T}_1, \ldots, \mathcal{T}_n$ et l'ensemble des transitions le produit des ensembles de transitions. Toutefois, afin de tenir compte des interactions entre les processus, on se donne une *contrainte de synchronisation*, c'est-à-dire un ensemble de n-uplets d'étiquettes, et on impose que les transitions du produit soient étiquetées par les éléments de la contrainte. Cette définition suppose que tous les processus du système exécutent leurs transitions simultanément, autrement dit que c'est la même horloge qui pilote tous les processus. On peut néanmoins simuler grâce à ce produit des systèmes asynchrones en considérant une action « nulle » qui représente le cas où un processus ne fait rien.

Dans de nombreux cas réalistes, les processus considérés ne peuvent être représentés que par des systèmes de transitions dont le nombre d'états est immense. Ce nombre peut parfois atteindre des tailles astronomiques. Comme nous l'avons noté plus haut, l'utilisation de systèmes de transitions infinis présentés de façon finie et compacte pourrait donner naissance à des techniques de conception et de vérification permettant d'éviter le problème de stockage des états.

L'étude des graphes infinis n'a décollé que récemment et un certain nombre de classes dont les éléments présentent des propriétés intéressantes ont été dégagées. Les premiers résultats relatifs aux liens entre la logique et les graphes infinis remontent aux travaux de J. Richard Büchi [Büc60] sur les mots infinis et de Michael Rabin sur les arbres binaires infinis [Rab69]. Ils ont en effet montré la décidabilité de la valeur de vérité de toute formule monadique du second ordre sur ces structures. Le résultat de Rabin sur les arbres binaires infinis revêt une importance capitale car il a permis de montrer la décidabilité de la théorie monadique du second ordre des autres classes de graphes qui ont depuis été étudiées. David Muller et Paul Schupp ont montré ce résultat pour les graphes associés aux machines à pile [MS85], puis Bruno Courcelle pour les graphes équationnels [Cou89] et enfin Didier Caucal pour les graphes préfixe-reconnaissables [Cau96].

Une conséquence du résultat de Muller et Schupp est la décidabilité du model checking sur les graphes de machines à pile pour le μ-calcul modal qui est une logique temporelle puissante. La procédure de décision qu'ils donnent originellement n'est cependant pas applicable aux cas pratiques. Olaf Burkart et Bernhard Steffen ont donné dans [BS92, BS94] un algorithme exponentiel de model checking sur ces graphes qui concerne une partie du μ-calcul modal. D'autre part, Igor Walukiewicz montre dans [Wal96] que le problème du model checking sur les graphes de machines à pile pour le μ-calcul modal entier est DEXPTIME-complet.

Enfin, notons que les systèmes temps-réel sont modélisés par des automates temporisés [AD90] dont le comportement, en général, ne peut être représenté que par un système de transitions infini. Des travaux ont été menés dans le but de vérifier des systèmes temps-réel ce qui a conduit à la mise au point d'outils logiciels. Uppaal (voir par exemple [LPY95] et [LPY97]) fait partie de ces outils ; il implémente un algorithme de model checking pour les automates temporisés et pour une logique temporisée particulière. Cet algorithme consiste à se ramener toujours à un ensemble fini d'états symboliques. Il est également possible grâce à ce logiciel d'effectuer de la vérification sur des automates hybrides linéaires (ceux-ci formant une extension des automates temporisés) simples.

Objectif de ce travail

L'objectif de ce travail est l'étude de quelques classes de graphes infinis associés à des machines de Turing et à des spécifications de Thue, c'est-à-dire en gros à des systèmes de réécriture de mots.

La classe des graphes finis est liée aux langages rationnels et celle des graphes de machines à pile aux langages algébriques. Ces deux classes de graphes ont déjà été étudiées mais on ne sait pratiquement rien au sujet des graphes liés aux niveaux suivants de la hiérarchie de Chomsky, c'est-à-dire les langages sensibles au contexte et les langages récursivement énumérables. Notre intérêt est donc de dégager un certain nombre de propriétés de ces graphes en nous concentrant sur des aspects liés à la spécification et à la vérification de processus.

Les spécifications de Thue ont été introduites par Teodor Knapik dans [Kna96] dans le but de mettre en place une technique de spécification et de vérification de processus ayant un nombre infini d'états. L'attrait des spécifications de Thue repose sur le fait qu'elles permettent d'exprimer des propriétés comme l'absence d'interblocage de manière syntaxique en termes de la théorie des langages et des systèmes de Thue [Thu14]. D'autre part, elles laissent envisager l'utilisation de techniques de démonstration automatique et de réécriture de mots pour vérifier ces propriétés.

Dans ce document, nous nous intéressons particulièrement au problème de la spécification de processus communiquants grâce aux graphes des machines de Turing et des spécifications de Thue. Nous considérons principalement les propriétés de fermeture de ces deux classes de graphes par produit synchronisé. Nous souhaitons ainsi contribuer au développement des connaissances concernant le produit synchronisé de graphes infinis dont on ne sait pratiquement rien actuellement.

Nous nous intéressons également au problème de la vérification formelle sur les graphes de ces deux classes. Nous montrons l'indécidabilité de la théorie du premier ordre des graphes des machines de Turing, des machines linéairement bornées et des spécifications de Thue. Nous insistons sur le fait que les graphes des machines de Turing que nous considérons sont constitués des sommets qui sont accessibles à partir d'une configuration distinguée appelée configuration initiale. Dans le cas où l'on s'intéresse au graphe constitué de toutes les configurations possibles de la machine, nous n'avons pas de réponse au problème de la décidabilité de la théorie du premier ordre. Nous examinons également la question de la conservation de la décidabilité de la théorie du premier ordre de graphes par produit synchronisé. Nous distinguons deux cas : celui du produit synchronisé défini par Arnold et Nivat et celui où l'on se restreint à la partie de ce produit qui est accessible depuis un sommet formé des racines des graphes que l'on considère. Dans le premier cas, nous montrons que si la théorie du premier ordre des graphes dont on fait le produit est décidable, alors la théorie du premier ordre du produit est, elle aussi, décidable.

Nous étudions enfin les liens qui unissent la classe des graphes des machines de Turing à celle des graphes des spécifications de Thue. Cette réflexion nous amène à considérer quelques sous-classes constituées de machines de Turing ou de spécifications de Thue particulières. Nous mettons en évidence l'existence de relations d'ε-équivalence entre les graphes de certaines de ces sous-classes.

Organisation de ce document

Ce document est composé de six chapitres sur le thème du produit synchronisé de graphes infinis. Les trois premiers chapitres situent le contexte de notre étude et rappellent les principaux résultats relatifs au domaine considéré. Les trois autres présentent les travaux qui ont été menés dans le cadre de cette thèse. Voici le contenu de chacune de ces parties.

Le premier chapitre traite des différents langages qui ont été utilisés pour exprimer et vérifier des propriétés de programmes informatiques. Ces langages sont des logiques dont plusieurs types existent et qui se regroupent en grandes catégories. Nous considérons la logique du premier ordre, la logique monadique du second ordre et enfin la logique temporelle. Chacune a des applications pratiques que nous présentons après avoir rappelé quelques résultats théoriques sur lesquels elles s'appuient. La logique temporelle est probablement celle qui a été le plus utilisée. Nous décrivons les différents types de systèmes de logique temporelle qui ont été étudiés et les propriétés que l'on peut exprimer au moyen de ces formalismes. Nous présentons enfin brièvement une technique de model checking dans le cas de la logique temporelle CTL.

Dans le second chapitre, nous introduisons les objets sur lesquels se concentre notre étude, c'est-à-dire les graphes infinis. Afin de pouvoir être utilisés informatiquement, ces graphes doivent être décrits de façon finie. Nous présentons les diverses descriptions finies qui ont été introduites et qui définissent des classes distinctes de graphes. Cette présentation tient également lieu d'historique très succinct de l'étude des graphes infinis. Les propriétés des graphes peuvent être exprimées au moyen de logiques spécialement adaptées. Nous rappelons dans un premier temps la façon dont un graphe peut être représenté par des structures relationnelles puis nous décrivons diverses logiques interprétées sur ces structures. Enfin, nous donnons la définition formelle du produit synchronisé de graphes en distinguant nettement le cas où les graphes dont on fait le produit ont une racine, c'est-à-dire un sommet distingué à partir duquel on peut accéder à tous les autres sommets.

Le troisième chapitre s'intéresse de façon détaillée aux classes de graphes infinis qui ont été étudiées jusqu'à aujourd'hui. Ces classes forment une hiérarchie pour l'inclusion au sommet de laquelle se trouvent les graphes rationnels. Ces graphes sont peu connus actuellement et seul un article très récent de Christophe Morvan [Mor00], que nous résumons, présente des résultats à leur sujet. La classe des graphes préfixe-reconnaissables a été introduite et étudiée par Didier Caucal [Cau96]. Nous rappelons les différentes définitions de ces graphes et leurs principales propriétés concernant le produit synchronisé et la décidabilité de la théorie monadique du second ordre. Une présentation identique est accordée aux graphes équationnels introduits par Bruno Courcelle [Cou89] et aux graphes des machines à pile introduits par David Muller et Paul Schupp [MS85] en indiquant toutefois quelques caractérisations connues de chacun d'eux dans la classe des graphes du niveau supérieur de la hiérarchie. Ce chapitre clôt la partie de cette thèse consacrée à l'état de l'art.

Le quatrième chapitre introduit les travaux que nous avons réalisés dans le cadre de notre thèse. Il est consacré aux graphes des spécifications de Thue. Nous définissons dans un premier temps ces spécifications et les graphes qui leur sont associés. Puis, nous décrivons quelques liens qui unissent ces graphes aux graphes des machines à pile, aux graphes préfixe-reconnaissables et aux graphes rationnels. Nous étudions ensuite la fermeture de la classe des graphes des spécifications de Thue par produit synchronisé. Nous construisons, à partir de spécifications données et d'une contrainte de synchronisation, une spécification de Thue dont le graphe est isomorphe au produit synchronisé des graphes des spécifications de départ. Nous nous intéressons également à la complexité de cette construction et nous terminons ce chapitre par un exemple complet destiné à améliorer la compréhension du lecteur.

Le cinquième chapitre est consacré aux graphes des machines de Turing. Nous précisons dans un premier temps la notion de machine de Turing que nous considérons puis nous décrivons la construction d'une machine à une seule bande de travail à partir d'une machine à plusieurs bandes. Cette construction aboutit à une machine dont le graphe est équivalent à celui de la machine de départ par rapport à une relation appelée ε-équivalence et qui est une équivalence observationnelle. Nous concentrons ensuite notre étude sur la fermeture de cette classe de graphes par produit synchronisé. La propriété de fermeture s'obtient assez facilement en construisant une machine dont le graphe est isomorphe au produit de celles dont on part. La partie la plus volumineuse de ce chapitre est consacrée aux liens qui unissent les graphes des machines de Turing à ceux des spécifications de Thue. Nous construisons à partir d'une spécification quelconque une machine de Turing dont le graphe est ε-équivalent à celui de la spécification. Nous montrons ensuite qu'une construction dans le sens inverse n'est pas toujours possible et nous donnons des conditions suffisantes qui permettent une telle construction. Enfin, nous terminons ce chapitre par l'étude de la décidabilité de la théorie du premier ordre des graphes des machines de Turing. Nous illustrons chaque construction réalisée dans ce chapitre au moyen d'exemples complets et nous signalons la validité de plusieurs résultats dans le cas des machines de Turing linéairement bornées.

Le sixième chapitre concerne la conservation de la décidabilité de la théorie du premier ordre par produit synchronisé. Nous montrons que les résultats obtenus dans le cas du produit synchronisé usuel et dans le cas où l'on se restreint à la partie de ce produit accessible depuis un sommet distingué ne sont pas identiques. Le raisonnement qui concerne le produit synchronisé usuel est basé sur une conséquence d'un théorème plus général introduit par Solomon Feferman et Robert Vaught [FV59]. Nous concluons alors par une discussion destinée à mettre en évidence un certain nombre de conditions telles que le produit de graphes qui les vérifient est égal à sa partie accessible. La théorie du premier ordre du produit synchronisé de tels graphes est décidable si la théorie du premier ordre de chaque graphe composant le produit l'est.

Chapitre 1

Logiques pour la vérification

Sommaire

Afin d'exprimer et de vérifier des propriétés de circuits logiques ou de programmes informatiques, un nombre assez important de logiques ont été étudiées. Elles se distinguent en général par le type de structures sur lesquelles on les interprète (des mots, des arbres ou encore des graphes), par le type de propriétés qu'elles permettent d'exprimer, par le fait qu'on puisse les axiomatiser par un système formel ou par l'existence d'algorithmes qui permettent de vérifier la validité d'une formule dans une structure donnée. L'objectif de ce chapitre est de présenter et de comparer trois grandes familles de logiques et d'exposer rapidement leurs principales caractéristiques et les utilisations pratiques dont elles ont fait l'objet.

1 La logique classique du premier ordre

La logique classique du premier ordre a surtout été utilisée en vérification sous sa forme équationnelle dans des méthodes basées sur les spécifications algébriques.

Depuis leur introduction [GTW78], ces dernières ont été largement reconnues comme adéquates pour le développement formel, la génération automatique de jeux de tests, la réutilisation et le prototypage. Toutefois, étant orientées vers les données, les aspects statiques et fonctionnels, elles s'avèrent insuffisantes pour traiter les problèmes liés à la concurrence et au parallélisme. De nombreuses recherches ont porté sur l'intégration des résultats du domaine du parallélisme à des méthodes de spécification orientées vers la modélisation des données, ce qui a donné naissance à de nouvelles approches (voir par exemple [AR91] pour un exposé de ces approches). Celles-ci possèdent de nombreux avantages en tant que techniques de spécification, mais, à notre connaissance, n'ont toujours pas donné de réponse aisée aux problèmes de vérification de propriétés dynamiques.

Les propriétés dynamiques sont souvent difficiles à exprimer au moyen des approches exposées dans [AR91] et on ne sait pas quelles sont les restrictions à imposer pour qu'elles soient décidables et de complexité pratique réaliste. De plus, lorsqu'on arrive à les exprimer, on se trouve confronté au problème délicat de les vérifier de façon automatisée. Nous ne connaissons pas de démonstrateur automatique permettant de vérifier des propriétés dynamiques qui est capable de construire ses preuves avec peu d'intervention humaine. Néanmoins, certaines propriétés dynamiques de sûreté (comme l'absence de blocage) peuvent être exprimées comme propriétés inductives du premier ordre ([Kna97] fournit quelques idées allant dans ce sens).

Les techniques de vérification de propriétés inductives ont beaucoup progressé depuis [AR91] grâce notamment à l'utilisation de la logique équationnelle avec appartenance (membership equational logic). Cette logique est assez simple et puissante. Ses formules atomiques sont des équations et des assertions stipulant l'appartenance à telle ou telle sorte et ses énoncés sont des clauses de Horn. Dans [BJM97], Adel Bouhoula, Jean Pierre Jouannaud et José Meseguer proposent l'utilisation de cette logique pour exprimer et vérifier des propriétés inductives. Ils étudient également des techniques de preuve pour la vérification de spécifications écrites en logique équationnelle avec appartenance. Ces techniques permettent en particulier de réaliser des preuves inductives.

2 La logique monadique du second ordre

La logique du second ordre est connue pour être très expressive mais également très complexe et peu de propriétés générales ont à l'heure actuelle été démontrées. Ses formules sont construites comme celles de la logique du premier ordre en considérant un type de variables supplémentaire appelées *variables du second ordre*. Ces variables sont interprétées par des relations et des fonctions d'arité quelconque et elles peuvent être quantifiées. Nous donnons, au paragraphe 4.3 du chapitre 2, des exemples de formules du second ordre et du second ordre monadique qui peuvent être interprétées sur des graphes.

Il existe plusieurs restrictions de la logique du second ordre conduisant à des systèmes dont le pouvoir d'expression est satisfaisant et qui possèdent de nombreuses propriétés de décidabilité. La logique *monadique* du second ordre et la logique du second ordre *faible* sont des exemples de telles restrictions. D'une part, elles n'autorisent aucun symbole de variable fonctionnelle. D'autre part, dans la première, les variables relationnelles représentent des relations monadiques, *i.e.* des ensembles, et dans la deuxième, ces variables représentent des relations finies.

Plusieurs systèmes de logique monadique du second ordre ont été développés, chacun utilisant une syntaxe et des structures d'interprétation particulières. La *logique monadique du second ordre avec un successeur*, notée S1S, fait partie de ces systèmes. Les formules de S1S sont interprétées sur des mots, finis ou infinis, construits sur un alphabet fini donné Σ. Tout mot α est représenté par une structure de la forme $(\mathbb{N}, (Q_a)_{a\in\Sigma})$ où $\mathbb{N}$ est l'ensemble des entiers naturels et, pour tout $a \in \Sigma$, $Q_a = \{n \in \mathbb{N} \mid \alpha(n) = a\}$ est l'ensemble des positions des caractères de α égaux à a (on suppose que $\alpha(0)$ est le premier caractère de α). Les termes de S1S sont construits à partir de la constante 0 et des variables d'objets x, y, $\ldots$ par application de la fonction successeur « $+1$ ». Les formules atomiques ont la forme $t = t'$, $t < t'$, $t \in X$, $t \in Q_a$ (pour tout $a \in \Sigma$) où t et t' sont des termes et X est une variable d'ensemble. Enfin, les formules de S1S sont construites à partir des formules atomiques et des connecteurs logiques et des quantificateurs classiques (la quantification est autorisée sur les variables d'objet et d'ensemble). Les variables d'objet sont interprétées par des éléments de $\mathbb{N}$ représentant des positions dans les mots et les variables d'ensemble par des sous-ensembles de $\mathbb{N}$. On associe à toute formule φ de S1S le langage $L(\varphi)$ des mots qui valident φ. On dit qu'un langage $\mathscr{L}$ est définissable en S1S s'il existe une formule φ telle que $\mathscr{L} = L(\varphi)$. Le système S1S est caractérisé par le résultat suivant.

Théorème de Büchi (cf. [Büc60]) *Un langage (de mots finis ou infinis) est définissable en S1S si et seulement s'il est régulier.*

La démonstration de ce théorème dans le sens gauche-droite repose sur la construction d'un automate qui reconnaît $L(\varphi)$ où φ est une formule de S1S. Cette construction est effective et est faite inductivement à partir de la structure de φ. De façon générale, on peut donc décider si un mot w valide une formule φ de S1S en construisant un automate $\mathcal{A}$ dont le langage reconnu est $L(\varphi)$ et en vérifiant si $w \in L(\varphi)$. On peut également vérifier si une formule φ est vraie sur un système de transitions fini en construisant un automate qui reconnaît $L(\neg\varphi)$, en faisant le produit de cet automate et du système de transitions et en vérifiant si le langage reconnu par l'automate obtenu est vide. En effet, d'une part le produit de deux automates reconnaît l'intersection des langages reconnus par les automates et, d'autre part, on peut décider si le langage reconnu par un automate de Büchi est vide (voir par exemple [Tho90]).

Le système WS1S (*logique monadique du second ordre faible avec un successeur*) est obtenu à partir de S1S en interprétant les symboles de variables d'ensembles uniquement par des sous-ensembles finis de $\mathbb{N}$. Le théorème de McNaughton [McN66] permet de démontrer que S1S et WS1S ont le même pouvoir d'expression (cf. [Tho90]).

Le système S2S (*logique monadique du second ordre avec deux successeurs*) est une extension de S1S. Les formules sont interprétées sur des arbres binaires finis ou infinis à valeurs dans un alphabet Σ, *i.e.* sur des applications $\mathcal{T} : \mathcal{D}\mathrm{om}(\mathcal{T}) \to \Sigma$ où $\mathcal{D}\mathrm{om}(\mathcal{T}) \subseteq \{0,1\}^*$ est un ensemble non-vide fermé par préfixe et vérifiant : $wj \in \mathcal{D}\mathrm{om}(\mathcal{T}), i < j \Rightarrow wi \in \mathcal{D}\mathrm{om}(\mathcal{T})$. Un arbre binaire $\mathcal{T}$ est représenté par une structure de la forme

$$\big(\{0,1\}^*, \varepsilon, \mathrm{succ}_0, \mathrm{succ}_1, <, (P_a)_{a\in\Sigma}\big)$$

où ε est le mot vide, succ_0 et succ_1 sont les deux fonctions successeur sur $\{0,1\}^*$ (avec $\mathrm{succ}_0(w) = w0$ et $\mathrm{succ}_1(w) = w1$), $<$ est la relation « est préfixe-propre de » et, pour tout $a \in \Sigma$, $P_a \subseteq \{0,1\}^*$ vérifie : $w \in P_a$ si et seulement si $\mathcal{T}(w) = a$. Les formules de S2S sont construites comme celles de S1S en utilisant, à la place de $+1$, les fonctions succ_0 et succ_1 pour construire les termes. Les variables d'objet représentent des éléments de $\{0,1\}^*$ et les variables d'ensemble des sous-ensembles de $\{0,1\}^*$.

Le système WS2S est obtenu à partir de S2S en interprétant les symboles de variables d'ensembles uniquement par des sous-ensembles finis. Contrairement au cas des systèmes S1S et WS1S, S2S est strictement plus expressif que WS2S. Le théorème de Büchi se généralise de la façon suivante.

Théorème 1.1 ([Don70, TW68]) *Un ensemble d'arbres finis est définissable en WS2S si et seulement s'il est reconnaissable par un automate d'arbre déterministe et bottom-up.*

Théorème 1.2 ([Rab69]) *Un ensemble d'arbres infinis est définissable en S2S si et seulement s'il est reconnaissable par un automate d'arbre de Rabin.*

Comme dans le cas de S1S, on peut construire, à partir d'une formule φ de S2S, un automate qui reconnaît les arbres qui valident φ. On peut donc décider si un arbre t valide une formule φ de S2S en construisant un automate $\mathcal{A}$ dont le langage reconnu est $L(\varphi)$ et en vérifiant si $t \in L(\varphi)$.

Des systèmes de logique monadique du second ordre dont les formules sont interprétées sur des graphes ont également été étudiés. En général, on distingue les logiques des graphes qui n'autorisent que la quantification sur les sommets et celles qui autorisent également la quantification sur les arcs. Nous donnons, au chapitre 2, la définition précise de la logique MSOL1 (quantification sur les sommets uniquement) et MSOL2 (quantification sur les sommets et les arcs). Notons qu'il est possible de traduire tout graphe $\mathcal{G}$ appartenant à certaines classes particulières de graphes et toute formule φ de MSOL1 en un arbre $\mathcal{T}$ et une formule φ' de S2S vérifiant : φ est valide dans $\mathcal{G}$ si et seulement si φ' est valide dans $\mathcal{T}$. Didier Caucal a proposé dans [Cau96] une telle traduction pour la classe des graphes préfixes-reconnaissables et a ainsi prouvé que l'on peut décider si un graphe quelconque de cette classe valide une formule donnée de MSOL1. Cependant, lorsque l'on désire raisonner sur des graphes, le système MSOL1 est bien entendu plus commode que S2S car il permet d'exprimer directement des propriétés de ces objets. En ce qui concerne la logique MSOL2, on ne connaît pas de transformation « simple » de ce style. Bruno Courcelle [Cou89] démontre que la théorie MSOL2 des graphes équationnels est décidable en transformant ces graphes en arbres binaires et en se ramenant au théorème de Rabin mais les transformations qu'il définit sont relativement complexes.

Les procédures de décision pour WS1S et WS2S ont été implémentées dans l'outil logiciel Mona [HJJ+95] (pour MONAdic second-order logic) qui a été développé au BRICS et à l'Université d'Aarhus au Danemark. Toutefois, à l'origine, les procédures programmées étaient celles de M2L(Str) qui est une logique monadique du second ordre sur les mots finis moins expressive que WS1S. Cet outil a été couplé à un autre logiciel, appelé Fido, qui fournit un formalisme logique qui combine WS1S et WS2S à des concepts de programmation tels que les enregistrements, les types de données récursifs et l'unification.

L'outil Mona traduit toute formule de WS1S ou WS2S en un automate fini qui reconnaît les mots ou les arbres qui valident la formule. Cette traduction est faite inductivement et nécessite une représentation et une gestion efficace des automates et des opérations qui leur sont appliquées (complémentation, produit, minimisation, ...). Cela est réalisé grâce à des diagrammes de décision binaires [Bry86] spécialement adaptés aux automates [Kla98].

L'objectif du projet Mona est d'exploiter le lien qui existe entre la logique et les automates et de montrer que la logique monadique du second ordre peut aider, de façon pratique, à identifier et à utiliser la régularité. Les applications de ce logiciel sont nombreuses et vont du traitement de textes à la vérification de systèmes distribués paramétrés. Il a ainsi permis de vérifier des systèmes distribués paramétrés (*i.e.* qui font intervenir un nombre de processus non fixé *a priori*) dont la correction avait jusqu'alors été prouvée par des méthodes lourdes d'induction semi-automatisées. Une version non triviale du problème des philosophes, proposée et vérifiée inductivement par Kurshan et McMillan dans [KM89], a pu être traitée efficacement et automatiquement grâce à Mona (voir [HJJ+95]).

Un autre logiciel, nommé MoSeL (pour MOnadic SEcond-order Logic), a également été développé à l'Université de Dortmund en Allemagne [KMMG97]. Actuellement, il implémente les procédures de décision pour M2L(Str), mais, grâce à son architecture modulaire, il peut être facilement étendu pour inclure des procédures de décision pour d'autres logiques comme par exemple WS1S ou WS2S. L'objectif du projet MoSeL est de montrer que la logique monadique du second ordre sur les mots fournit une théorie attrayante pour raisonner sur les structures paramétrées. Le système M2L(Str) est considéré comme un langage de spécification abstrait et comme un langage de programmation efficace qui permet de vérifier de manière automatique des systèmes paramétrés. L'accent est donc surtout mis sur l'analyse et la vérification en logique monadique du second ordre.

3 La logique temporelle

La logique temporelle est une logique monomodale, *i.e.* une logique modale qui ne traite que d'un seul concept, le temps. Elle permet, grâce aux opérateurs modaux, de raisonner sur des assertions à un point du temps.

Amir Pnueli dans [Pnu77] fut l'un des premiers à remarquer que la logique temporelle est un formalisme particulièrement bien adapté à la spécification et à la vérification de programmes informatiques *réactifs* [Pnu86]. De tels programmes, de façon idéale, ne s'arrêtent jamais. De plus, ils reçoivent des entrées et produisent des sorties en maintenant une interaction permanente avec leur environnement. Du fait de cette interaction, toute sortie intermédiaire peut influencer une entrée future, ce qui amène à dire que la classe des programmes réactifs subsume celle des programmes concurrents. Les systèmes d'exploitation ou les protocoles de communication réseau constituent des exemples de programmes réactifs.

Les programmes réactifs se distinguent des programmes séquentiels ordinaires. Ces derniers, après avoir calculé un résultat, s'arrêtent. La correction d'un programme ordinaire peut se formuler en termes de précondition et de postcondition dans un formalisme tel que la logique de Hoare car la sémantique sous-jacente peut être définie comme une transformation à partir d'un état initial jusqu'à un état final. Lorsqu'il n'y a pas d'état final, comme dans le cas des programmes réactifs, cette façon de procéder ne s'applique pas. Par contre, l'utilisation des opérateurs de la logique temporelle est bien appropriée à la description du comportement des programmes réactifs à travers le temps.

Dans la suite, lorsque nous parlerons de *programme*, nous entendrons *programme informatique réactif*.

3.1 Classification des systèmes de la logique temporelle

La majeure partie des activités de recherche sur la logique temporelle s'est concentrée sur les systèmes globaux, basés sur des points du temps, à temps discret et à opérateurs sur le futur uniquement. Cependant, afin de donner au lecteur une idée du large éventail des possibilités de définition d'un système de logique temporelle, nous décrivons ci-après les diverses alternatives.

Systèmes propositionnels, systèmes du premier ordre

En logique temporelle propositionnelle, la partie non temporelle de la logique est simplement une logique propositionnelle classique. Les formules sont donc construites à partir des symboles de constantes *vrai* et *faux*, de symboles de variables propositionnelles, de connecteurs logiques et d'opérateurs modaux. La logique temporelle propositionnelle correspond au niveau de raisonnement le plus abstrait.

Les formules de la logique temporelle du premier ordre sont construites à partir de symboles de constantes, de symboles de variables individuelles, de symboles de fonctions, de symboles de prédicats et des quantificateurs $\exists$ et $\forall$.

Systèmes globaux, systèmes modulaires

Dans un système de logique temporelle *global*, tous les opérateurs modaux sont interprétés dans un unique univers correspondant à un programme. Ces systèmes conviennent bien au raisonnement global sur un programme complet. Dans un système de logique temporelle *modulaire*, la syntaxe des opérateurs modaux permet d'exprimer des propriétés de correction concernant plusieurs programmes différents au sein d'une même formule. De tels systèmes permettent de vérifier un programme en s'intéressant à chacun de ses sous-programmes [BKP84, Pnu84].

Systèmes à temps discret, systèmes à temps continu

La plupart des systèmes de logique temporelle utilisés pour raisonner sur des programmes utilisent un temps *discret*. Dans ces systèmes, le présent correspond à l'état actuel du programme et le futur correspond à l'état successeur immédiat. Les philosophes ont également étudié des systèmes temporels interprétés par un temps *continu*. Une application de ces systèmes au raisonnement sur les programmes concurrents a été proposée dans [BKP86].

Points et intervalles

La majeure partie des logiques temporelles utilisées pour raisonner sur des programmes est basée sur des opérateurs qui sont évalués en des *points* du temps. Certains formalismes (cf. [HS91], [SMSV83] et [Mos83]) ont toutefois été définis au moyen d'opérateurs modaux qui sont évalués sur des *intervalles* de temps. L'argument des défenseurs de cette dernière méthode consiste à dire que l'utilisation des intervalles de temps facilite grandement la formulation de certaines propriétés de correction.

Systèmes avec et sans opérateurs sur le passé

Parmi les modalités temporelles d'origine créées par les philosophes, on trouve des opérateurs qui décrivent des événements aussi bien dans le futur que dans le passé. Cependant, dans les logiques temporelles utilisées pour raisonner sur les programmes, seuls des opérateurs sur le futur sont fournis. On peut en effet montrer, comme conséquence du fait que les programmes ont un temps d'exécution initial, que l'inclusion d'opérateurs sur le passé n'ajoute aucun pouvoir expressif. Cependant, certains ont avancé (cf. [LPZ85]) que l'ajout de tels opérateurs rendait plus naturelle et plus commode la formulation de spécifications.

Systèmes à temps linéaire, systèmes à temps arborescent

Lorsque l'on définit un système de logique temporelle, on peut considérer deux natures différentes du temps. La première se distingue par la *linéarité* du cours du temps, *i.e.* à chaque moment il n'y a qu'un seul futur possible. L'autre se caractérise par l'*arborescence* du temps, *i.e.* à chaque moment il y a plusieurs futurs alternatifs possibles. Dans un système de logique temporelle à temps linéaire, la sémantique est définie par une structure de Kripke dont la relation d'accessibilité est un ordre total. Dans un système de logique temporelle à temps arborescent, la sémantique est définie par une structure de Kripke dont la relation d'accessibilité peut être représentée par un arbre. Dans certains cas, la structure du temps est quelconque, mais les opérateurs temporels fournis ainsi que la définition de la relation de satisfaction correspondent à une considération linéaire ou arborescente du temps.

Les systèmes de logique temporelle linéaire globaux, à temps discret et à opérateurs sur le futur uniquement ont été souvent utilisés en vérification. Ils proposent en général les opérateurs temporels **F**, **G**, **N** et **U** qui se lisent, respectivement, *à un moment du futur*, *toujours dans le futur*, *à l'instant suivant* (ou *next*) et *jusqu'à ce que* (ou *until*). De tels systèmes sont intéressants car ils représentent bien les situations suivantes. D'une part, l'horloge des ordinateurs actuels détermine un temps ponctuel et discret. De plus, tout programme réactif tourne depuis un instant initial et, de façon idéale, éternellement dans le futur. Enfin, il est parfois utile, pour vérifier un programme, de s'intéresser à son évolution au cours du temps en examinant la suite des états qu'il prend lorsqu'il parcourt un chemin bien précis. Les formules bien formées des logiques linéaires employées sont donc interprétées par un ensemble d'instants du temps en bijection avec l'ensemble des entiers naturels $\mathbb{N}$.

La logique temporelle linéaire propositionnelle (PLTL) et la logique du premier ordre sur les mots ont le même pouvoir d'expression [Kam68, GPSS80]. La première peut, en effet, être interprétée sur les mots finis ou infinis. Les formules de la logique sont alors construites à partir des symboles de variables propositionnelles notés p_a pour tout a appartenant à un alphabet donné Σ et la relation de satisfaction est définie ainsi (α est un mot, fini ou infini, construit sur Σ et φ est une formule) :

- $\alpha \models p_a$ ssi $\alpha(0) = a$,
- $\alpha \models \mathbf{N}\varphi$ ssi $\alpha(1)\alpha(2)\cdots \models \varphi$,
- $\alpha \models \mathbf{F}\varphi$ ssi $\exists i \geq 0,\ \alpha(i)\alpha(i+1)\cdots \models \varphi$,
- $\alpha \models \mathbf{G}\varphi$ ssi $\forall i \geq 0,\ \alpha(i)\alpha(i+1)\cdots \models \varphi$,
- $\alpha \models \varphi\mathbf{U}\psi$ ssi $\exists i \geq 0,\ \alpha(i)\alpha(i+1)\cdots \models \psi$ et $\forall 0 \leq j < i,\ \alpha(j)\alpha(j+1)\cdots \models \varphi$.

La logique du premier ordre sur les mots est, quant à elle, définie comme la logique S1S (voir le paragraphe 2 de ce chapitre) sans considérer les variables du second ordre. On sait que tout langage de mots finis ou infinis est définissable en logique du premier ordre sur les mots si et seulement s'il est sans étoile [Lad77, Tho79, Tho81] (*i.e.* il peut être dénoté par une expression régulière construite à partir de sous-ensembles de l'alphabet avec un nombre fini de concaténations, d'unions et de complémentations). Par conséquent, d'après le théorème de Büchi, le pouvoir d'expression de S1S est supérieur à celui de PLTL. Dans [Wol83], Wolper a proposé une extension de PLTL, appelée ETL pour Extended Temporal Logic, qui définit exactement les ensembles réguliers de mots finis ou infinis.

La logique CTL, pour Computation Tree Logic, est une logique temporelle propositionnelle à temps arborescent qui fut proposée par Clarke et Emerson dans [CE81] dans le but de vérifier des programmes informatiques réactifs. Les structures d'interprétation des formules de CTL ne définissent pas forcément directement un arbre. Le terme « temps arborescent » provient en fait des remarques suivantes. D'une part, les formules de CTL sont interprétées dans des structures telles qu'à chaque instant présent correspondent plusieurs futurs possibles. D'autre part, la logique propose les quantificateurs **E** (pour *il existe un futur*) et **A** (pour *pour tous les futurs*) qui permettent de tenir compte de l'existence de ces futurs. De plus, bien qu'un instant présent puisse avoir plusieurs passés possibles dans les structures d'interprétation considérées, CTL ne fournit pas d'opérateurs permettant de considérer ces passés. Enfin, un instant présent survient après passage par un unique passé. Par conséquent, bien que les formules soient interprétées dans des structures quelconques, cela revient, vu la définition de la logique, à déplier la structure en un arbre, d'où l'adjectif « arborescent » pour qualifier le temps de CTL.

Les formules bien formées de CTL sont construites de telle sorte que les quantificateurs **E** et **A** doivent obligatoirement être suivis d'un opérateur temporel de la logique temporelle linéaire (ils s'emploient donc uniquement sous la forme **EF**, etc.) Cette restriction n'est plus imposée dans le cas de la logique temporelle CTL* [EH86] qui constitue une extension de CTL et qui subsume la logique temporelle linéaire.

3.2 Propriétés vérifiées grâce à la logique temporelle

La logique classique du premier ordre n'est pas suffisante lorsque l'on désire exprimer des propriétés qui sont liées à l'évolution d'un programme au cours du temps ou aux activités possibles d'un programme. Ces propriétés font en général partie des deux grandes familles suivantes.

Propriétés de vivacité (*liveness properties*). Elles traduisent le fait que quelque chose de bon se passera finalement dans le programme. Par exemple, le programme réagira finalement à une stimulation ou réagira finalement après avoir atteint une certaine situation. Les propriétés d'*équité* (*fairness properties*) font partie de cette classe. Ces dernières garantissent que le choix répété entre deux activités alternatives d'un programme est juste (aucune activité n'est choisie pour toujours). L'absence de famine est un exemple de propriété d'équité.

Propriétés de sûreté (*safety properties*). Elles traduisent le fait que rien de mauvais ne se produira dans le programme. Par exemple, après avoir atteint telle situation, aucune situation incorrecte ne sera atteinte. Parmi les exemples célèbres de propriétés de sûreté, notons la propriété d'exclusion mutuelle pour l'accès à une section critique ainsi que l'absence d'impasses (*deadlock*).

Sous l'impulsion d'Amir Pnueli [Pnu77], de nombreuses logiques temporelles ont été étudiées et utilisées pour exprimer et vérifier de telles propriétés. Des travaux ont également été menés pour caractériser ces propriétés de façon syntaxique et sémantique (cf. [AS85, Sis85]) ou alors topologique (cf. [AS87, LPZ85, MP89]). En ce qui concerne les propriétés de vivacité, plusieurs caractérisations topologiques ont été proposées. Alpern et Schneider ont par exemple suggéré de les considérer comme des ensembles denses de la topologie de Cantor [AS87]. Les propriétés de sûreté sont en général caractérisées par des ensembles fermés de la topologie de Cantor.

Voici quelques exemples d'expression de propriétés de vivacité et de sûreté au moyen de la logique temporelle linéaire. L'absence de famine peut se formuler par $\mathbf{G}[\mathrm{essaieSC}_i \Rightarrow \mathbf{F}\ \mathrm{enSC}_i]$, ce qui signifie qu'à chaque fois que le processus i essaie d'accéder à la section critique, finalement, il y parvient. La propriété d'exclusion mutuelle dans le cas de deux processus peut se formuler ainsi : $\mathbf{G}[\neg[\mathrm{enSC}_1 \wedge \mathrm{enSC}_2]]$. Enfin, la formule $\mathbf{G}[\neg\mathrm{bloqué}_1 \wedge \cdots \wedge \neg\mathrm{bloqué}_m]$ traduit l'absence d'impasses dans un programme faisant intervenir m processus.

Dans le cas d'une logique à temps arborescent comme CTL, les notions d'inévitabilité et de potentialité sont exprimables ce qui permet, contrairement à la logique linéaire, de formuler des propriétés comme la présence d'une impasse dans un certain futur. Par contre, CTL est limitée par le fait qu'elle ne permet pas d'exprimer des propriétés d'équité. En effet, ces dernières s'écrivent généralement sous la forme $\mathbf{AGF}\varphi$ qui signifie que φ est vraie infiniment souvent dans tous les futurs possibles. Or, une telle formule ne vérifie pas la syntaxe de CTL. Cette limitation de CTL a conduit à la définition de la logique FairCTL [EL87] qui permet d'exprimer les propriétés d'équité.

3.3 Les méthodes de preuve

Ces méthodes consistent à exprimer un modèle du programme à vérifier ainsi que les propriétés désirées dans une logique convenable puis à prouver que le modèle possède ces propriétés. La preuve est construite à partir des axiomes et des règles d'inférence d'un système formel qui axiomatise la sémantique considérée. Cette approche est très puissante et flexible, mais la construction des preuves est souvent lourde et difficile. Cependant, l'intervention de l'Homme permet en général d'anticiper, grâce à l'intuition, un schéma de preuve efficace.

Dans [MP83], Manna et Pnueli proposent, sous la forme d'un système formel et de techniques de preuve, un système de preuve complet basé sur la logique temporelle linéaire. Ce système permet de vérifier des programmes concurrents et est constitué de trois parties distinctes composées de schémas d'axiomes et de règles d'inférences qui peuvent être directement adaptés au langage de programmation considéré. La première partie fournit des axiomes et des règles d'inférence généraux pour la logique temporelle du premier ordre avec l'égalité. La seconde permet de raisonner sur les variables et les structures de données. Les axiomes et règles fournis peuvent être adaptés pour traiter des domaines spécifiques tels que les entiers, les chaînes de caractères, les listes, etc. Enfin, la troisième partie est spécialement dédiée au raisonnement sur les programmes. Elle permet de vérifier que le programme considéré fonctionne correctement par rapport à des critères d'initialisation et d'équité qui dépendent du langage utilisé.

Dans [EH85], Emerson et Halpern proposent un ensemble de schémas d'axiomes et de règles d'inférence qui forment un système de déduction correct et complet pour CTL.

3.4 Le model checking

De façon générale, le problème du model checking se formule ainsi : étant données une structure temporelle finie Ω et une formule φ d'une logique temporelle propositionnelle, est-ce que φ est valide dans Ω ?

Le terme « model checking » fut inventé par Clarke et Emerson (cf. [CE81]) lorsqu'ils présentèrent un algorithme efficace de vérification de modèle pour la logique CTL et qu'ils proposèrent d'utiliser cet algorithme comme base d'une technique de vérification automatique de programmes. Queille et Sifakis dans [QS82] proposèrent eux aussi, à peu près à la même époque, un algorithme de model checking pour une logique arborescente similaire.

Le model checking a été abondamment utilisé pour vérifier des programmes. Dans ce cadre, la structure temporelle qui est considérée est un modèle d'un programme ayant un nombre fini d'états. Un état d'un programme est alors représenté par un instant du temps. La formule φ dont on veut vérifier la validité dans la structure temporelle est une spécification de la correction du programme. L'avantage de cette approche est qu'elle permet d'engendrer des algorithmes rapides et qu'elle est entièrement automatisable. De plus, les logiques utilisées autorisent l'expression et la vérification d'une grande variété de propriétés, notamment des propriétés de sûreté, de vivacité et d'équité.

Un grand nombre de recherches théoriques ont été menées dans le but de classifier les différentes logiques temporelles par rapport à la complexité du model checking qu'elles engendrent. Les résultats obtenus pour les logiques que nous avons présentées sont les suivants.

Théorème 1.3 (Sistla et Clarke [SC85])

- *Le problème du model checking pour la logique temporelle linéaire propositionnelle est PSPACE-complet.*
- *Le problème du model checking pour la logique temporelle linéaire propositionnelle avec l'opérateur* **F** *uniquement est NP-complet.*

Théorème 1.4 (Wolper [Wol86])

Le problème du model checking pour CTL est en temps déterministe polynomial.

En ce qui concerne le model checking basé sur CTL, les algorithmes efficaces sont obtenus grâce à des méthodes de calcul de points fixes. Clarke et Emerson ont en effet montré que les opérateurs de cette logique peuvent être caractérisés comme des points fixes de fonctions monotones [CE81]. Dans le théorème qui suit, $\mu y.f$ (resp. $\nu y.f$) dénote le plus petit (resp. le plus grand) point fixe de la fonction f de la variable y. De plus, les formules de CTL sont identifiées avec l'ensemble des instants de la structure temporelle considérée dans lesquels la formule est vraie. Chaque membre des égalités suivantes représente donc des ensembles d'instants.

Théorème 1.5 (Clarke et Emerson) *Pour toute structure temporelle arborescente et pour toutes variables propositionnelles p et q,*

1. $\mathbf{EF}p = \mu y.(p \vee \mathbf{EN}y)$,
2. $\mathbf{EG}p = \nu y.(p \wedge \mathbf{EN}y)$,
3. $\mathbf{E}(p\mathbf{U}q) = \mu y.(q \vee (p \wedge \mathbf{EN}y))$.

Le calcul des points fixes dans une structure temporelle dont l'ensemble des instants est dénoté par M s'effectue au moyen de l'algorithme suivant (en convenant que pour toute formule φ, M_φ dénote l'ensemble des instants de M où φ est vraie et que $\perp$ dénote la constante *faux* et $\top$ la constante *vrai*).

Pour calculer le plus petit (resp. le plus grand) point fixe de la fonction f :
$\varphi := \perp$ (resp. $\varphi := \top$) ;
Faire
$\varphi' := \varphi$;
$\varphi := f(\varphi)$;
jusqu'à ce que $M_{\varphi'} = M_\varphi$;
Renvoyer M_φ.

Voici un petit exemple de calcul de point fixe utilisant cet algorithme. On considère la structure temporelle suivante où la variable propositionnelle p est vraie à l'instant 2.

$$\begin{tikzcd} 1 \arrow[r] & \overset{p}{2} \arrow[r] & 3 \arrow[r] \arrow[ll, bend left] & 4 \end{tikzcd}$$

Calculons l'ensemble des instants de cette structure dans lesquels la formule $\mathbf{EF}p$ est vraie. D'après le théorème 1.5, cet ensemble est le plus petit point fixe de la fonction $f : y \mapsto p \vee \mathbf{EN}y$. Voici les résultats obtenus après chaque itération de l'algorithme :

1. $\varphi' = \bot$, $\varphi = f(\bot) = p \vee \mathbf{EN}\bot$, $M_\varphi = M_p = \{2\}$ et $M_{\varphi'} = \varnothing$,
2. $\varphi' = f(\bot)$, $\varphi = f(f(\bot))$, $M_\varphi = \{1, 2\}$ et $M_{\varphi'} = \{2\}$,
3. $\varphi' = f(f(\bot))$, $\varphi = f(f(f(\bot)))$, $M_\varphi = \{1, 2, 3\}$ et $M_{\varphi'} = \{1, 2\}$,
4. $\varphi' = f(f(f(\bot)))$, $\varphi = f(f(f(f(\bot))))$, $M_\varphi = \{1, 2, 3\}$ et $M_{\varphi'} = \{1, 2, 3\}$.

L'algorithme s'arrête donc à la quatrième itération et renvoie l'ensemble d'instants $\{1, 2, 3\}$.

L'inconvénient majeur de cet algorithme est, qu'avant de l'appliquer, il est nécessaire de construire le graphe d'état complet du programme modélisé. Or, ce graphe d'état est souvent très vaste, ce qui conduit irrémédiablement à ce que l'on appelle communément le problème de *l'explosion du nombre des états*. Kenneth McMillan a toutefois montré dans sa thèse [McM92] que ce problème peut être évité si l'on représente le modèle implicitement au moyen d'une formule booléenne. Son approche, baptisée *model checking symbolique*, permet en particulier la manipulation des modèles grâce à des *diagrammes de décision binaires* [Bry86] dont la gestion est connue pour être efficace. Ce travail a permis la mise au point d'un système de vérification appelé SMV [McM92]. Ce système fournit un langage de description qui peut être annoté par des spécifications exprimées par des formules de CTL. Il fournit également un vérificateur qui extrait un système de transitions de tout programme écrit en SMV et qui prouve que ce système est ou n'est pas un modèle des spécifications écrites en CTL.

Néanmoins, SMV ne manipule que des systèmes de transitions ayant un nombre fini d'états. Dans le cas de programmes concurrents qui font intervenir des données, cette technique de vérification ne peut donc être appliquée car le nombre d'états des systèmes de transitions qui interviennent est souvent infini.

4 Le μ-calcul

Le μ-calcul a été introduit par D. Scott et J. W. De Bakker [SB69] et développé par plusieurs chercheurs, notamment D. Park [Par70, Par74], W. P. De Roever [Roe74] et V. Pratt [Pra81]. Il peut être vu comme une extension d'une logique par des opérateurs de plus petit et de plus grand point fixe. Il existe donc plusieurs systèmes de μ-calcul, chacun étant basé sur une logique particulière.

Le μ-calcul permet d'exprimer de nombreuses propriétés concernant les états d'un programme comme par exemple le fait de mener à une impasse, le fait d'être accessible ou encore d'être équivalent à un autre état. Son pouvoir d'expression est strictement plus grand que celui de CTL et que celui de la Logique Dynamique Propositionnelle (PDL). Park a introduit dans [Par74] un μ-calcul qui a été utilisé par exemple dans [BCM$^+$90] pour vérifier des propriétés de programmes comme celles mentionnées ci-dessus. L'algorithme utilisé repose sur une technique de model checking efficace basée sur les diagrammes de décision binaires ordonnés.

Le μ-calcul de Park est *relationnel* dans le sens où les opérateurs de point fixe sont appliqués à des symboles de relations. Il permet par exemple de calculer l'ensemble des états d'un système de transitions menant à une impasse. Si l'on note G la relation de transition du système, cet ensemble peut être calculé à partir de la formule $\mu Z.\lambda(x)\big[\forall y.G(x,y) \Rightarrow Z(y)\big]$ qui exprime le fait que la relation unaire Z est la plus petite solution de l'équation $\{x \mid Z(x)\} = \{x \mid \forall y.G(x,y) \Rightarrow Z(y)\}$ (Z est donc un plus petit point fixe).

Le langage Toupie [Rau95] a été la première implémentation complète du μ-calcul sur les domaines finis. Il permet de formuler et de résoudre des contraintes sur des domaines finis dans lesquelles les relations peuvent être exprimées comme point fixe d'équations. Il implémente un algorithme de model checking et utilise les diagrammes de décision binaires [Bry86] pour représenter les relations et les formules. Toupie permet donc de modéliser et de résoudre efficacement des problèmes tels que la vérification de propriétés de machines ayant un nombre fini d'états.

Chapitre 2

Les graphes infinis, leurs théories, leur produit synchronisé

Sommaire

Nous introduisons dans ce chapitre les objets sur lesquels nous centrons notre étude, à savoir les graphes infinis. Contrairement au cas des graphes finis qui ont déjà bénéficié d'une étude approfondie, l'engouement des chercheurs pour ce type de graphe est assez récent. Nous proposons ci-après un bref historique des études qui leur ont été consacrées. Nous nous intéressons également à quelques logiques de graphes dont la présentation vient compléter l'exposé qui a été réalisé au chapitre 1. Enfin, nous introduisons le produit synchronisé de graphes qui fut, comme chacun le sait, défini par André Arnold et Maurice Nivat dans le but de représenter des systèmes de processus interagissant.

1 Notations générales

Tout au long de ce document, nous employons les notations générales suivantes. L'ensemble des entiers naturels est noté $\mathbb{N}$ et, si $n \in \mathbb{N}$, $[n]$ dénote l'ensemble $\{1, \ldots, n\}$ (avec $[0] = \varnothing$). De plus, le domaine d'une relation binaire $\mathbf{R}$ est noté $\mathcal{D}\mathrm{om}(\mathbf{R})$ et son image $\mathcal{R}\mathrm{an}(\mathbf{R})$. Enfin, si $\vec{t}$ est un n-uplet, $\pi_i(\vec{t})$ dénote sa i^{e} coordonnée.

Nous supposons que le lecteur possède une bonne connaissance des définitions et résultats fondamentaux de la théorie des langages. Les notations que nous utilisons lorsque nous considérons des alphabets, des langages et des mots sont les suivantes. Tout d'abord, nous travaillons uniquement avec des alphabets *finis*, généralement dénotés par les lettres Σ ou Γ. Nous notons Σ^* le monoïde libre engendré par l'alphabet fini Σ. L'élément neutre de ce monoïde est le mot vide noté ε et Σ^+ représente l'ensemble $\Sigma^* \setminus \{\varepsilon\}$. La *taille* de Σ est le nombre de lettres qu'il contient et est notée $|\Sigma|$. La *longueur* d'un mot w construit sur Σ est notée $|w|$. Pour tout $i \in [\,|w|\,]$, on note $w(i)$ le i^{e} caractère de w.

Nous considérerons parfois des alphabets étendus par le mot vide. Nous introduisons donc dès maintenant la notation suivante : pour tout alphabet Σ, le symbole $\widetilde{\Sigma}$ représente l'ensemble $\Sigma \cup \{\varepsilon\}$.

2 Graphes

2.1 Définitions

Il existe plusieurs notions de graphes. Un graphe peut avoir des arcs orientés, des arcs étiquetés, il peut être un hypergraphe (*i.e.* ses arcs peuvent relier plusieurs sommets à la fois), il peut également être simple (*i.e.* entre deux sommets, il y a au plus un seul arc, la notion d'arc dépendant bien sûr du fait que le graphe est orienté ou pas, étiqueté ou pas).

Les graphes que nous considérons dans ce document sont orientés, étiquetés par un alphabet fini et simples dans le sens où entre deux sommets x et y, pour un a donné, il y a au plus un seul arc qui va de x à y et qui est étiqueté par a. Ces graphes ne sont pas des hypergraphes, c'est-à-dire que leurs arcs ne relient que deux sommets. De plus, ils sont généralement infinis, c'est-à-dire que leur ensemble de sommets est infini. Pour être plus précis, voici des définitions formelles.

Un *graphe* $\mathcal{G}$ étiqueté par un alphabet fini Σ est un ensemble de *sommets* V et d'*arcs* E tels que $E \subseteq V \times \Sigma \times V$. On dit que $\mathcal{G}$ est *infini* lorsque son ensemble de sommets est infini. Étant donnés v et v' dans V, un arc de v à v' étiqueté par $a \in \Sigma$ est noté $v \xrightarrow[\mathcal{G}]{a} v'$. Par conséquent, $\xrightarrow[\mathcal{G}]{a}$ est une relation binaire sur V pour chaque $a \in \Sigma$. Un *chemin* de v à v' est une suite d'arcs de la forme $v_1 \xrightarrow[\mathcal{G}]{a_1} v_2, \ldots, v_i \xrightarrow[\mathcal{G}]{a_i} v_{i+1}, \ldots, v_{n-1} \xrightarrow[\mathcal{G}]{a_{n-1}} v_n$ telle que $n \in \mathbb{N}$, $v_0 = v$ et $v_n = v'$ (le chemin vide est donc autorisé). Le mot $w = a_0 \ldots a_n$ est l'*étiquette* du chemin et on écrit $v \xrightarrow[\mathcal{G}]{w} v'$.

Les graphes $\mathcal{G}$ que nous considérons sont parfois étiquetés par $\widetilde{\Sigma}$, *i.e.* par les éléments d'un alphabet fini et par le mot vide. Dans un tel cas, nous utilisons la notation et la définition suivantes : la fermeture réflexive et transitive de $\xrightarrow[\mathcal{G}]{\varepsilon}$ est notée $\overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}}$ et on pose, pour tout $a \in \Sigma$, $\overset{a}{\underset{\mathcal{G}}{\dashrightarrow}} = \overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}} \circ \xrightarrow[\mathcal{G}]{a} \circ \overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}}$.

Un sommet r est une *racine* d'un graphe $\mathcal{G}$ si, pour tout sommet de $\mathcal{G}$, il existe un chemin de r jusqu'à ce sommet. Un graphe pouvant avoir plusieurs racines, il est parfois utile d'en distinguer une par rapport aux autres ; on exprime le fait que r est une racine de $\mathcal{G}$ que l'on désire distinguer par la notation $(\mathcal{G}, r)$.

2.2 Isomorphisme et ε-équivalence

Soit $\mathcal{G}_1$ et $\mathcal{G}_2$ deux graphes ayant pour ensemble de sommets V_1 et V_2 et étiquetés par $\widetilde{\Sigma_1}$ et $\widetilde{\Sigma_2}$ respectivement.

On dit que $\mathcal{G}_1$ et $\mathcal{G}_2$ sont *isomorphes* lorsqu'il existe deux bijections, f de V_1 dans V_2 et g de $\widetilde{\Sigma_1}$ dans $\widetilde{\Sigma_2}$, telles que (v_1, a, v_1') est un arc de $\mathcal{G}_1$ si et seulement si $(f(v_1), g(a), f(v_1'))$ est un arc de $\mathcal{G}_2$. Dans le cas où l'on distingue une racine, *i.e.* dans le cas où l'on considère des couples $(\mathcal{G}_1, r_1)$ et $(\mathcal{G}_2, r_2)$, on impose également que $f(r_1) = r_2$.

On dit que les graphes $(\mathcal{G}_1, r_1)$ et $(\mathcal{G}_2, r_2)$ sont *ε-équivalents* s'il existe une relation $\leftrightsquigarrow \subseteq V_1 \times V_2$ telle que

1. $\mathcal{D}\mathrm{om}(\leftrightsquigarrow) = V_1$ et $\mathcal{R}\mathrm{an}(\leftrightsquigarrow) = V_2$;
2. $r_1 \leftrightsquigarrow r_2$;
3. pour tout a dans Σ, tous sommets v_1 et v_1' de $\mathcal{G}_1$ tels que $v_1 \overset{a}{\underset{\mathcal{G}_1}{\dashrightarrow}} v_1'$ et tout sommet v_2 de $\mathcal{G}_2$ tel que $v_1 \leftrightsquigarrow v_2$, il existe un sommet v_2' de $\mathcal{G}_2$ tel que $v_1' \leftrightsquigarrow v_2'$ et $v_2 \overset{a}{\underset{\mathcal{G}_2}{\dashrightarrow}} v_2'$;
4. pour tout a dans Σ, tous sommets v_2 et v_2' de $\mathcal{G}_2$ tels que $v_2 \overset{a}{\underset{\mathcal{G}_2}{\dashrightarrow}} v_2'$ et tout sommet v_1 de $\mathcal{G}_1$ tel que $v_1 \leftrightsquigarrow v_2$, il existe un sommet v_1' de $\mathcal{G}_1$ tel que $v_1' \leftrightsquigarrow v_2'$ et $v_1 \overset{a}{\underset{\mathcal{G}_1}{\dashrightarrow}} v_1'$.

Notons que l'ε-équivalence est une équivalence observationnelle (également appelée *bisimulation faible*) selon la définition de Robin Milner dans [Mil80] si l'on considère ε comme étant un événement non-observable.

3 Présentations des graphes infinis

Comme nous l'avons mentionné plus haut, les graphes que nous considérons sont en général infinis. Alors que les graphes finis ont été largement étudiés, les graphes infinis n'ont attiré l'attention des chercheurs que récemment. D'un point de vue informatique, une *présentation*, c'est-à-dire une description, finie de ces graphes devient nécessaire dès qu'on désire les étudier. On s'intéresse alors aux graphes qui sont associés à des objets mathématiques finis particuliers et on étudie les propriétés de ces graphes en se basant sur celles des objets.

L'étude des graphes de Cayley de groupes context-free a conduit David Muller et Paul Schupp à associer à tout automate à pile un graphe de transition dont les sommets sont les configurations internes de l'automate [MS85, MS81, MS83]. Ils appellent les graphes ainsi obtenus *graphes context-free*. Une extension des graphes de Cayley à la présentation de monoïdes est proposée dans [KC99]. Elle consiste en la donnée d'un système de réécriture de mots et d'un sous-ensemble rationnel de l'ensemble des mots irréductibles. Nous détaillons cette présentation dans le chapitre 4 de ce document.

Bruno Courcelle a également introduit une présentation qui consiste à considérer des systèmes d'équations (voir par exemple [Cou89]). Il a appelé *équationnels* les hypergraphes qui sont obtenus en considérant la plus petite solution de systèmes d'équations faisant intervenir des opérations de remplacement d'hyperarcs (opérations *HR* en abrégé). La classe des graphes qui sont présentés de cette manière contient strictement celle des graphes de David Muller et Paul Schupp. Ce résultat, d'abord conjecturé par Bruno Courcelle dans [Cou89], a été prouvé par Michel Bauderon dans [Bau92]. De plus, les systèmes d'équations peuvent être considérés comme des grammaires d'hypergraphes à remplacement d'hyperarcs et déterministes (ou grammaires HR déterministes, voir par exemple [Cou90a]).

C'est sous cet angle « grammatical » que Didier Caucal a étudié cette classe qu'il baptise plutôt classe des graphes *réguliers* [Cau92]. Par la suite, il a défini dans [Cau96] la classe des graphes *préfixe-reconnaissables* présentés au moyen d'extensions de relations reconnaissables ou de la composition de deux opérations de la théorie des langages appliquée à l'arbre binaire complet. Il a également montré dans [Cau96] que la classe des graphes préfixe-reconnaissables est plus générale que celle des graphes réguliers.

Klaus Barthelmann a finalement prouvé dans [Bar97] que la classe des graphes préfixe-reconnaissables est la même que celle des graphes qui sont obtenus en considérant la plus petite solution de systèmes d'équations faisant intervenir des opérations de remplacement de sommets (opérations *VR* en abrégé) [ER91]. Les graphes de cette classe sont donc eux aussi équationnels, mais plus par rapport aux opérations HR et, afin d'éviter toute confusion, Barthelmann utilise dans [Bar98] les termes *HR-équationnels* et *VR-équationnels* pour bien distinguer les graphes dont on parle.

Plusieurs terminologies, faisant référence à des présentations distinctes, existent donc actuellement pour désigner les mêmes classes de graphes infinis. Par souci d'homogénéité, nous n'en utiliserons qu'une seule le long de ce document, celle de Didier Caucal. Cette terminologie permet de différencier clairement les graphes équationnels HR et VR. Nous parlerons donc de graphes réguliers ou de graphes préfixe-reconnaissables plutôt que de graphes équationnels comme Bruno Courcelle ou de graphes HR-équationnels ou VR-équationnels comme Klaus Barthelmann.

Comme le remarquent Didier Caucal et Teodor Knapik dans [CK99a], les présentations que nous avons mentionnées peuvent être classées en deux catégories, celle où les graphes sont définis à isomorphisme près et celle où les sommets des graphes sont définis explicitement. La première catégorie est celle des présentations *externes* et la deuxième celle des présentations *internes*. On peut donc dire que les présentations de [Cou89, Bar97] sont externes, alors que celles de [MS85, KC99, Hun98] sont internes. Dans ces listes de références, [Hun98] présente des travaux qui ont conduit à la définition d'une présentation basée sur des systèmes d'équations avec une récursion d'ordre supérieur (voir aussi [Hun99]). Enfin, la présentation détaillée dans [Cau96] qui consiste à appliquer des opérations à l'arbre binaire complet est externe. Par contre, celle qui consiste en la considération de relations reconnaissables étendues est interne.

4 Quelques logiques de graphes

Nous avons introduit au chapitre 1 les logiques LTL, CTL et, très brièvement, CTL* qui sont des logiques de graphe.

Nous présentons maintenant la logique de Hennessy–Milner ainsi que la logique du premier ordre et du second ordre des graphes. Au cours des chapitres qui suivent, nous considérerons la décidabilité de ces logiques lorsque nous étudierons des classes de graphes particulières.

4.1 La logique de Hennessy–Milner

La logique de Hennessy–Milner [HM85] est une logique de graphes propositionnelle définie de la façon suivante.

Soit $\mathcal{G} = (V, E)$ un graphe étiqueté par l'alphabet Σ. Les formules de la logique sont formées à partir des symboles de constantes $vrai$ et $faux$, de symboles de variables propositionnelles p_0, $p_1, \ldots$, des connecteurs $\neg$, $\vee$ et $\wedge$ et des opérateurs unaires $\langle a \rangle$ pour tout $a \in \Sigma$. Les symboles de constantes et de variables sont des formules bien formées et si φ et ψ sont des formules bien formées alors $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$ et $\langle a \rangle \varphi$, pour tout $a \in \Sigma$, en sont aussi. Enfin, un sommet de $\mathcal{G}$ vérifie la formule $\langle a \rangle \varphi$ si et seulement s'il est l'origine d'un arc étiqueté par a dont l'extrémité vérifie φ.

Le pouvoir d'expression de la logique de Hennessy–Milner est moins fort que celui de la logique du premier ordre que nous présentons dans la suite de ce texte.

4.2 Structures relationnelles et logiques associées

Pour présenter les différentes logiques du premier et second ordre d'un graphe, nous introduisons la notion de structure relationnelle que nous utiliserons pour représenter un graphe.

Soit R un ensemble de symboles de relations tel que pour chaque r dans R, l'arité de r est notée $\rho(r)$. Une *structure relationnelle* sur R est un n-uplet

$$S = \big(D_S, (r_S)_{r \in R}\big)$$

où D_S est un ensemble appelé *domaine* de S et, pour chaque r dans R, r_S est un sous-ensemble de $D_S^{\rho(r)}$. Les propriétés de telles structures peuvent être décrites au moyen de formules de la logique du premier ordre, monadique du second ordre ou du second ordre.

Les formules de la *logique du premier ordre* sont écrites avec des variables d'objet notées x, $y, \ldots$ qui sont interprétées par des éléments de D_S. Les formules atomiques sont toutes les $x = y$ et les $r(x_1, \ldots, x_n)$ où $n = \rho(r)$. Les formules sont construites à partir des formules atomiques et des connecteurs logiques et quantificateurs classiques.

Les formules de la *logique du second ordre* sont construites avec, en plus des variables d'objets, des variables du second ordre notées X, Y, $\ldots$ qui sont interprétées par des relations d'arité quelconque sur D_S. Les formules atomiques sont donc aussi les $(x_1, \ldots, x_p) \in X$. Les formules sont construites à partir des formules atomiques et des connecteurs logiques et quantificateurs classiques.

Enfin, les formules de la *logique monadique du second ordre* sont construites comme celles de la logique de second ordre mais avec la contrainte suivante : les variables du second ordre ne dénotent que des relations unaires sur D_S, *i.e.* des sous-ensembles de D_S.

Il existe une extension de la logique monadique du second ordre appelée *logique monadique du second ordre avec comptage modulaire* (*counting monadic second-order logic*). Ses formules permettent de tester si un ensemble est fini et si le cardinal d'un ensemble est égal à n modulo p où n et p sont des entiers naturels tels que $0 \leq n < p$ et $p \geq 2$. Plus précisément, cette logique comporte deux types de formules atomiques supplémentaires qui ont la forme $\mathbf{fin}(X)$ et $\mathbf{card}_{n,p}(X)$. La première de ces deux formules est vraie sur S si et seulement si le sous-ensemble de D_S dénoté par X est fini et la seconde si et seulement si ce sous-ensemble est fini et a un cardinal égal à n modulo p.

L'ensemble des formules du premier ordre (resp. monadiques du second ordre, resp. du second ordre) sans variable libre valides dans S est appelé *théorie du premier ordre* (resp. *théorie monadique du second ordre*, resp. *théorie du second ordre*) de S. Une théorie est *décidable* si et seulement si l'appartenance d'une formule sans variable libre à cette théorie est décidable.

4.3 Représentation d'un graphe par une structure relationnelle

Un graphe simple, orienté et étiqueté peut être représenté par une structure relationnelle de plusieurs façons. On peut par exemple considérer que le domaine de la structure est l'ensemble des sommets du graphe et que les relations sont binaires et représentent l'existence d'un arc entre deux sommets. On peut également considérer que le domaine de la structure contient à la fois les sommets et les arcs du graphe. Dans ce cas, on peut aussi quantifier sur les arcs ou les ensembles d'arcs.

Ces deux manières de représenter un graphe, c'est-à-dire en considérant que le domaine de la structure contient ou non les arcs, ont été étudiées notamment dans [Cou94]. Il convient de bien les distinguer car certaines propriétés de graphe ne peuvent être exprimées sans la quantification sur les arcs.

À tout graphe $\mathcal{G} = (V, E)$ étiqueté par l'alphabet Σ et dont le sommet r est une racine distinguée, nous associons donc deux structures relationnelles. La première est une structure sur $R_1 = \{\mathbf{r}\} \cup \{\mathbf{arc}[a] \mid a \in \Sigma\}$ où $\mathbf{r}$ est un symbole de relation unaire et, pour tout $a \in \Sigma$, $\mathbf{arc}[a]$ est un symbole de relation binaire. Cette structure, notée $|\mathcal{G}|_1$, est définie ainsi : son domaine est l'ensemble V des sommets de $\mathcal{G}$, $\mathbf{r}_{|\mathcal{G}|_1} = \{r\}$ et pour tout $a \in \Sigma$, $\mathbf{arc}[a]_{|\mathcal{G}|_1} = \big\{(v, v') \in V^2 \mid v \xrightarrow[\mathcal{G}]{a} v'\big\}$.

La deuxième est une structure sur $R_2 = \{\mathbf{r}\} \cup \{\mathbf{arc}[a] \mid a \in \Sigma\}$ où $\mathbf{r}$ est un symbole de relation unaire et, pour tout $a \in \Sigma$, $\mathbf{arc}[a]$ est un symbole de relation ternaire. Cette structure, notée $|\mathcal{G}|_2$, est définie ainsi : son domaine est $V \cup E$, $\mathbf{r}_{|\mathcal{G}|_2} = \{r\}$ et pour tout $a \in \Sigma$, on a $\mathbf{arc}[a]_{|\mathcal{G}|_2} = \big\{(e, v, v') \mid e \in E \text{ et } e = (v, a, v')\big\}$.

Toute formule écrite dans une logique associée à $|\mathcal{G}|_1$ peut être réécrite en une formule équivalente de la même logique associée à $|\mathcal{G}|_2$. L'inverse n'est pas toujours vrai. Il suffit de considérer l'exemple suivant : le fait qu'un graphe a un circuit hamiltonien peut être exprimé par une formule monadique du second ordre avec quantification sur des ensembles d'arcs (voir par exemple [Cou90b]). Cette quantification est nécessaire et, pour le montrer, il suffit de considérer la classe $K_{n,m}$ des graphes bipartis ayant n sommets bleus, m sommets rouges et uniquement des arcs reliant tout sommet rouge à chaque sommet bleu et tout sommet bleu à chaque sommet rouge. Un graphe de $K_{n,m}$ a un circuit hamiltonien si et seulement si $n = m$. Si la propriété d'avoir un circuit hamiltonien était exprimable en logique monadique du second ordre sans quantification sur les arcs, alors on pourrait exprimer, sans quantification sur les arcs, le fait que deux ensembles ont le même cardinal. Or, cela n'est pas le cas, comme démontré dans [Cou90b].

Les logiques associées à $|\mathcal{G}|_2$ peuvent être comparées de la façon suivante. Pour commencer, la logique du premier ordre est assez faible dans la mesure où ses formules ne permettent d'exprimer que des propriétés locales de $\mathcal{G}$. C'est en effet ce qu'affirme le théorème de Gaifman [Gai82] :

Théorème 2.1 (Gaifman) *Une propriété de $\mathcal{G}$ est exprimable par une formule fermée de la logique du premier ordre si et seulement si elle est équivalente à une combinaison booléenne de propriétés de la forme :*

$$\exists\, x_1,\ldots,x_n\Big[\bigwedge\{P(N(x_i,m))\mid i\in[n]\}\wedge\bigwedge\{d(x_i,x_j)>2m\mid 1\leq i<j\leq n\}\Big]$$

*où $d(x,y)$ dénote la distance entre deux sommets (*i.e. *la longueur minimale d'un chemin de l'un à l'autre, les arcs pouvant être pris dans n'importe quel sens), m est un entier, $N(x,m)$ est le sous-graphe de $\mathcal{G}$ constitué de tous les sommets dont la distance à x est au plus m et des arcs qui les relient et P est une propriété exprimable en logique du premier ordre.*

Par exemple, les propriétés « $\mathcal{G}$ est k-coloriable » (pour un k fixé) et « $\mathcal{G}$ est hamiltonien » ne sont pas exprimables en logique du premier ordre. La logique monadique du second ordre est une extension de la logique du premier ordre et permet d'exprimer des propriétés intéressantes concernant des chemins. On peut par exemple exprimer la propriété d'être un arbre ou encore l'existence d'un chemin entre les sommets x et y dont toutes les étiquettes appartiennent à un ensemble donné. Cependant, quelques propriétés utiles de graphes ne sont pas exprimables en logique monadique du second ordre. C'est le cas par exemple de la propriété « dans $\mathcal{G}$, il y a autant d'arcs étiquetés par a que d'arcs étiquetés par b ». La raison est qu'en logique monadique du second ordre, il n'est pas possible de « compter ». Toutefois, cette logique dispose de plusieurs propriétés de décidabilité intéressantes qui ne sont pas valides en logique du second ordre. La logique monadique du second ordre avec comptage modulaire est strictement plus expressive que la logique monadique du second ordre [Cou90b, Cou89]. Enfin, la logique du second ordre est une extension de la logique monadique du second ordre et la plupart des propriétés des graphes sont exprimables par ses formules.

Voici, pour finir, quelques exemples de formules pour certaines logiques évoquées dans cette partie.

Exemple 2.2 *La formule*

$$\forall\, x\;\exists\, y\;\bigvee_{a\in\Sigma}\mathbf{arc}[a](x,y)$$

appartient à la théorie du premier ordre de $|\mathcal{G}|_1$ si et seulement si $\mathcal{G}$ n'a pas de sommets bloquants.

Exemple 2.3 *La formule*

$$\begin{aligned}\exists\, X,Y\,[\quad &\forall x(x\in X\vee x\in Y)\wedge\forall x\,\neg(x\in X\wedge x\in Y)\wedge\\ &\quad\forall\, x,y,z(\bigvee_{a\in\Sigma}\mathbf{arc}[a](z,x,y)\Rightarrow\neg(x\in X\wedge y\in X)\wedge\neg(x\in Y\wedge y\in Y)\quad]\end{aligned}$$

appartient à la théorie monadique du second ordre de $|\mathcal{G}|_2$ si et seulement si $\mathcal{G}$ est 2-coloriable.

Exemple 2.4 *Soit φ la formule*

$$\forall x,y,z\;[(x,y)\in X\wedge(x,z)\in X\Rightarrow y=z]\;\wedge\;\forall x,y,z\;[(y,x)\in X\wedge(z,x)\in X\Rightarrow y=z]$$

exprimant le fait que la relation binaire X est une bijection. Alors,

$$\exists X \; [\quad \varphi \;\wedge\; \forall x,y,z\; (\mathbf{arc}[a](x,y,z) \Rightarrow \exists\, x'\; (x,x') \in X)$$

$$\wedge\; \forall x,y,z\; (\mathbf{arc}[b](x,y,z) \Rightarrow \exists\, x'\; (x',x) \in X) \qquad]$$

appartient à la théorie du second ordre de $|\mathcal{G}|_2$ si et seulement si dans $\mathcal{G}$ il y a autant d'arcs étiquetés par a que d'arcs étiquetés par b.

5 Produit synchronisé de graphes

Le produit synchronisé de graphes a été introduit par André Arnold et Maurice Nivat dans [AN82, Niv79]. C'est une opération fondamentale dans la définition de la sémantique d'un système de processus interagissant. Pour de plus amples informations, nous conseillons au lecteur l'étude de l'ouvrage [Arn92].

Étant donnés n alphabets $\Sigma_1, \ldots, \Sigma_n$, une *contrainte de synchronisation* $\mathscr{C}$ sur $\Sigma_1, \ldots, \Sigma_n$ est un sous-ensemble de $\prod_{i\in[n]} \Sigma_i$. Soit $\mathcal{G}_i = (V_i, E_i)$, $i \in [n]$, des graphes tels que pour chaque $i \in [n]$, $\mathcal{G}_i$ est étiqueté par Σ_i.

Le *produit synchronisé* des $\mathcal{G}_i$ par rapport à la contrainte $\mathscr{C}$ est noté $\prod_{i\in[n]}^{\mathscr{C}} \mathcal{G}_i$ et est le graphe étiqueté par $\mathscr{C}$ dont l'ensemble des sommets est $V = \prod_{i\in[n]} V_i$ et l'ensemble d'arcs est

$$\left\{ (v,c,v') \in V \times \mathscr{C} \times V \;\middle|\; \forall i \in [n],\; \pi_i(v) \xrightarrow[\mathcal{G}_i]{\pi_i(c)} \pi_i(v') \right\} .$$

Lorsque $\mathscr{C} = \prod_{i\in[n]} \Sigma_i$, le produit synchronisé est aussi appelé *produit libre* car toutes les combinaisons d'étiquettes sont autorisées.

Enfin, dans le cas où les graphes $\mathcal{G}_i$ ont une racine distinguée r_i, nous considérons la définition suivante qui est une variante de celle d'Arnold et Nivat. Soit Π le sous-graphe maximal du produit défini ci-dessus dont $(r_1, \ldots, r_n)$ est une racine. Le produit synchronisé des graphes $(\mathcal{G}_i, r_i)$ est le graphe $\big(\Pi, (r_1, \ldots, r_n)\big)$.

Chapitre 3

Produit synchronisé de quelques sous-classes des graphes rationnels

Sommaire

Les graphes rationnels ont été introduits récemment par Christophe Morvan dans [Mor00]. Nous résumons ci-après les résultats présentés dans cet article. Certaines sous-classes de celle des graphes rationnels ont déjà fait l'objet d'une étude approfondie et de nombreux résultats les concernant ont été démontrés. Nous présentons dans ce chapitre toutes ces sous-classes ainsi que les principales propriétés qui les caractérisent.

1 Les graphes rationnels

On considère deux alphabets finis Z et Σ.

Définition 3.1 *Un graphe $\mathcal{G}$ dont les sommets sont des éléments de Z^* et qui est étiqueté par Σ est* rationnel *lorsque, pour tout $a \in \Sigma$, $\underset{\mathcal{G}}{\xrightarrow{a}}$ est une relation rationnelle.*

Par conséquent, pour tout $a \in \Sigma$, $\xrightarrow[\mathcal{G}]{a}$ est reconnue par un transducteur rationnel.

Les caractérisations suivantes sont données dans [Mor00].

1. *Caractérisation interne.* La relation $\{(w, a, w') \mid w \xrightarrow[\mathcal{G}]{a} w'\}$ appartient au plus petit sous-ensemble de l'ensemble des parties de $Z^* \times \Sigma \times Z^*$ qui contient $\varnothing$, $\{(\varepsilon, a, \varepsilon) \mid a \in \Sigma\}$, $\{(\varepsilon, a, z) \mid a \in \Sigma,\ z \in Z\}$ et $\{(z, a, \varepsilon) \mid a \in \Sigma,\ z \in Z\}$ et qui est fermé par union, concaténation et $+$. Ces deux dernières opérations sont définies ainsi. Pour tout $A, B \subseteq Z^* \times \Sigma \times Z^*$, $A.B = \{(uu', a, vv') \mid (u, a, v) \in A \text{ et } (u', a, v') \in B\}$ et $A^+ = \bigcup_{i \geq 1} A^i$ avec $A^1 = A$ et $A^{i+1} = A^i.A$.

2. *Caractérisation externe.* Nous rappelons tout d'abord au lecteur qu'un langage est linéaire s'il est engendré par une grammaire algébrique dont les règles de production ne contiennent au plus qu'un non-terminal dans leur membre droit. Un graphe est rationnel si et seulement s'il peut être obtenu à partir de l'arbre binaire complet au moyen d'une substitution linéaire inverse un peu particulière suivie d'une restriction rationnelle. Nous invitons le lecteur à se rendre au paragraphe 2.1 de ce chapitre où les notions d'arbres binaire complet, de restriction et de substitution sont définies précisément. Notons qu'une substitution est linéaire si elle associe à tout élément de son ensemble de départ un langage linéaire.

La classe des graphes rationnels est fermée par produit synchronisé. En effet, soit $\mathcal{G}_1$ et $\mathcal{G}_2$ deux graphes rationnels étiquetés par Σ_1 et Σ_2 respectivement. Soit $\mathscr{C}$ une contrainte de synchronisation sur Σ_1 et Σ_2. Soit $(a_1, a_2) \in \mathscr{C}$ et $\mathcal{T}_{a_1}$ et $\mathcal{T}_{a_2}$ deux transducteurs reconnaissant $\xrightarrow[\mathcal{G}_1]{a_1}$ et $\xrightarrow[\mathcal{G}_2]{a_2}$ respectivement. Alors, $\xrightarrow[\mathcal{G}]{(a_1,a_2)}$, où $\mathcal{G}$ est le produit synchronisé de $\mathcal{G}_1$ et $\mathcal{G}_2$ selon $\mathscr{C}$, est la relation reconnue par le transducteur associé au produit libre des graphes de $\mathcal{T}_{a_1}$ et $\mathcal{T}_{a_2}$.

Enfin, pour tout graphe rationnel $\mathcal{G}$, la théorie du premier ordre de $|\mathcal{G}|_1$ n'est pas décidable [Mor00]. On peut néanmoins décider si $\mathcal{G}$ est déterministe ou si deux graphes rationnels déterministes sont égaux ou si l'un est inclus dans l'autre.

2 Les graphes préfixe-reconnaissables

La présentation des éléments de cette classe au moyen de relations préfixe-reconnaissables est due à Didier Caucal [Cau96]. Comme nous l'avons écrit au chapitre 2, c'est sa terminologie que nous utilisons le long de ce document. Toutefois, les graphes préfixe-reconnaissables peuvent également être présentés au moyen d'opérations sur l'arbre binaire complet [Cau96] ou de systèmes d'équations régulières [Bar97].

Dans la suite de ce paragraphe, nous détaillons chacune de ces présentations et nous nous intéressons à la théorie monadique du second ordre des graphes préfixe-reconnaissables ainsi qu'à leur produit synchronisé.

2.1 Les présentations des graphes préfixe-reconnaissables

Relations préfixe-reconnaissables

Soit Z un alphabet fini. Une relation binaire $\mathbf{R}$ sur Z^* est *préfixe-reconnaissable* si et seulement s'il existe un ensemble fini I et, pour tout $i \in I$, des sous-ensembles rationnels U_i, V_i et W_i de Z^* tels que

$$\mathbf{R} = \bigcup_{i \in I} (U_i \times V_i) W_i$$

où

$$(U_i \times V_i) W_i = \{(uw, vw) \mid u \in U_i, v \in V_i, w \in W_i\} .$$

L'ensemble des relations binaires préfixe-reconnaissables sur Z^* est une extension de l'ensemble des relations binaires reconnaissables sur Z^*. Ces dernières sont, en effet, caractérisées par le théorème de Mezei qui stipule que l'ensemble de ces relations est

$$\left\{ \bigcup_{i \in I} U_i \times V_i \;\middle|\; I \text{ est fini, } U_i \text{ et } V_i \text{ sont des sous-ensembles rationnels de } Z^* \right\} .$$

L'ensemble des relations préfixe-reconnaissables forme une algèbre de Boole [Cau96]. De plus, la fermeture réflexive et transitive d'une relation préfixe-reconnaissable est aussi une relation préfixe-reconnaissable [Cau96].

Un *graphe* $\mathcal{G}$ dont les sommets sont des éléments de Z^* et qui est étiqueté par Σ est *préfixe-reconnaissable* si, pour tout $a \in \Sigma$, $\xrightarrow[\mathcal{G}]{a}$ est une relation préfixe-reconnaissable.

Opérations sur l'arbre binaire complet

On considère les graphes obtenus en appliquant à l'arbre binaire complet Λ une substitution rationnelle inverse suivie d'une restriction rationnelle. Voici la définition de chacune de ces notions.

Arbre binaire complet sur l'alphabet $\{a, b\}$. C'est le graphe noté Λ, étiqueté par $\{a, b\}$, dont l'ensemble des sommets est $\{a, b\}^*$ et dont l'ensemble d'arcs est

$$\left\{ (w, \sigma, w') \in \{a, b\}^* \times \{a, b\} \times \{a, b\}^* \mid w' = w\sigma \right\} .$$

Restriction rationnelle. Soit $\mathcal{G} = (V, E)$ un graphe étiqueté par l'alphabet Σ et soit L un sous-ensemble de V. La restriction de $\mathcal{G}$ selon L est notée $\mathcal{G}_{|L}$ et est le graphe étiqueté par Σ dont l'ensemble des sommets est L et dont l'ensemble d'arcs est $\left\{ (v, \sigma, v') \in E \mid v, v' \in L \right\}$. Une restriction de $\mathcal{G}$ est rationnelle lorsque L est un langage rationnel.

Substitution rationnelle inverse. Étant donné un graphe $\mathcal{G} = (V, E)$ étiqueté par l'alphabet Σ, on introduit le nouvel alphabet $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$ qui va servir à prendre les arcs de $\mathcal{G}$ en sens inverse. On considère donc, pour chaque $a \in \Sigma$, les nouvelles relations $\xrightarrow[\mathcal{G}]{\overline{a}}$ définies comme suit :

$$\forall v, v', \; v \xrightarrow[\mathcal{G}]{\overline{a}} v' \iff v' \xrightarrow[\mathcal{G}]{a} v .$$

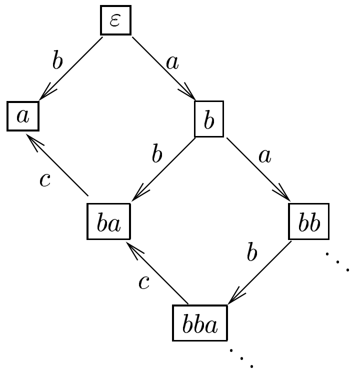
Une substitution h sur Σ^* est un morphisme de Σ^* dans l'ensemble des parties de $(\Sigma \cup \overline{\Sigma})^*$: $h(\varepsilon) = \{\varepsilon\}$ et pour tout w et w' dans Σ^*, $h(ww') = h(w)h(w')$. La substitution inverse $h^{-1}(\mathcal{G})$ du graphe $\mathcal{G}$ par rapport à h est le graphe étiqueté par Σ dont l'ensemble des sommets est V et l'ensemble des arcs est $\left\{(v, c, v') \in V \times \Sigma \times V \mid \exists w \in h(c),\ v \xrightarrow[\mathcal{G}]{w} v'\right\}$. La substitution inverse $h^{-1}(\mathcal{G})$ est rationnelle lorsque h prend ses valeurs dans l'ensemble des parties rationnelles de $(\Sigma \cup \overline{\Sigma})^*$.

On note $\mathrm{Rat}(E)$ l'ensemble des parties rationnelles de tout ensemble E. Étant donné un alphabet Σ contenant au moins les lettres a et b, la classe $\mathrm{REC}_{\mathrm{Rat}}(\Sigma)$ est définie comme suit :

$$\mathrm{REC}_{\mathrm{Rat}}(\Sigma) = \left\{ h^{-1}(\Lambda)_{|L} \;\middle|\; \begin{array}{l} h \ :\ \Sigma^* \to \mathrm{Rat}(\{a, b, \overline{a}, \overline{b}\}^*) \text{ est un morphisme et} \\ L \in \mathrm{Rat}(\{a, b\}^*) \end{array} \right\}.$$

Cette classe contient donc des graphes étiquetés par Σ. On définit alors $\mathrm{REC}_{\mathrm{Rat}}$ comme l'union des $\mathrm{REC}_{\mathrm{Rat}}(\Sigma)$ pour tous les alphabets Σ contenant au moins les lettres a et b.

Exemple 3.2 (Tiré de [Cau96]) *Soit Σ l'alphabet $\{a, b, c, d\}$, L le langage $\varepsilon + b^*a$ et h la substitution définie par $h(a) = \{b\}$, $h(b) = \{b\overline{b}a\}$, $h(c) = \{\overline{a}\overline{b}a\}$ et $h(d) = \varnothing$. Le graphe $h^{-1}(\Lambda)_{|L}$ est :*



Le résultat suivant est démontré dans [Cau96].

Théorème 3.3 *L'ensemble des graphes préfixe-reconnaissables forme un ensemble complet de représentants de* $\mathrm{REC}_{\mathrm{Rat}}$.

Systèmes d'équations avec opérations VR

Cette présentation fait intervenir la notion d'hypergraphe que nous définissons ci-après.

Soit L un ensemble fini d'étiquettes fixé pour toute cette partie et auquel est associée une application *rang* $\rho : L \to \mathbb{N} \setminus \{0\}$. Un *hypergraphe simple, orienté, étiqueté par L, avec sources et de sorte* $n \in \mathbb{N}$ consiste en la donnée

- d'un ensemble de *sommets* V,
- d'un ensemble d'*hyperarcs* $E \subseteq L \times V^*$,

– d'une application *source* $\mathrm{src} : [n] \to V$,

tels que pour tout $(l, w) \in E$, $|w| = \rho(l)$. Le sommet $\mathrm{src}(i)$ est appelé i-ième source de l'hypergraphe.

Voici à présent la définition des opérations VR telles que définies par Barthelmann dans [Bar97].

Composition parallèle. Soit n et n' deux entiers naturels non nuls. La composition parallèle $||_{n,n'}$ est une opération binaire définie comme suit. Soit $\mathcal{G}$ et $\mathcal{G}'$ deux hypergraphes de sorte n et n' et d'ensemble de sommets V et V' disjoints. Alors, $\mathcal{G}||_{n,n'}\mathcal{G}'$ est un graphe de sorte $\max(n, n')$. De plus, si $\approx$ est la plus petite relation d'équivalence sur $V \cup V'$ telle que, pour tout $i \in [\min(n, n')]$, $\mathrm{src}(i) \approx \mathrm{src}'(i)$, alors $\mathcal{G}||_{n,n'}\mathcal{G}'$ est défini ainsi :

– l'ensemble des sommets est $(V \cup V')/ \approx$,

– l'ensemble des arcs est $\{(l, [v_1]. \ldots .[v_m]) \mid (l, v_1 \ldots v_m) \in E \cup E'\}$,

– l'application source est $([\,] \circ \mathrm{src}) \cup ([\,] \circ \mathrm{src}')$.

Dans cette définition, $(V \cup V')/ \approx$ est l'ensemble des classes d'équivalence des éléments de $V \cup V'$ modulo $\approx$ et $[\,] : V \cup V' \to (V \cup V')/ \approx$ est la surjection qui, à tout élément de $V \cup V'$, associe sa classe d'équivalence modulo $\approx$.

Opérations positives sans quantificateur. Soit n un entier naturel non nul. On considère l'ensemble de symboles $R_n = \{\mathbf{s}_i \mid i \in [n]\} \cup \{\mathbf{arc}[l] \mid l \in L\}$ où les $\mathbf{s}_i$ sont des symboles de constante et les $\mathbf{arc}[l]$ des symboles de relation d'arité $\rho(l)$.

Soit $\vec{x}$ une suite finie de variables sans répétition. On considère les formules atomiques $M_1 = M_2$ et $\mathbf{arc}[l](M_1, \ldots, M_k)$ où les M_j sont soit des variables de $\vec{x}$, soit un $\mathbf{s}_i$. L'ensemble des formules positives et sans quantificateur sur n et $\vec{x}$ est noté $\mathrm{PSQ}_n(\vec{x})$ et est l'ensemble des formules construit à partir des formules atomiques, des symboles de constante *vrai* et *faux* et des connecteurs booléens $\wedge$ et $\vee$.

Soit m un entier naturel non nul. Un schéma de définition Δ positif, sans quantificateur et de n à m est la donnée :

– de spécifications de source $\sigma_i \in [n]$ pour chaque $i \in [m]$,

– d'une formule de sommet $\delta \in \mathrm{PSQ}_n(x_1)$ qui a la forme $\delta' \vee \bigvee_{i \in [m]} x_1 = s_{\sigma_i}$ où δ' est une formule quelconque,

– de formules d'hyperarcs $\eta_l \in \mathrm{PSQ}_n(x_1, \ldots, x_{\rho(l)})$ pour chaque $l \in L$.

Le schéma Δ détermine une opération unaire Def_Δ qui s'applique à tout hypergraphe $\mathcal{G}$ de sorte n. L'opération Def_Δ est une *opération positive sans quantificateur* et son résultat sur $\mathcal{G}$ est un hypergraphe de sorte m défini ainsi. Considérons que $\mathcal{G}$ est représenté par la structure relationnelle $|\mathcal{G}|$ sur R_n dont le domaine est l'ensemble V des sommets de $\mathcal{G}$ et telle que, pour tout $l \in L$,

$$\mathbf{arc}[l]_{|\mathcal{G}|} = \{(v_1, \ldots, v_{\rho(l)}) \mid (l, v_1, \ldots, v_{\rho(l)}) \text{ est un hyperarc de } \mathcal{G}\}$$

et, pour tout $i \in [n]$, $\mathbf{s}_{i|\mathcal{G}|}$ est la i-ième source de $\mathcal{G}$. En interprétant toute variable de $\vec{x}$ par un sommet de $\mathcal{G}$, le symbole *vrai* par V et le symbole *faux* par $\varnothing$, on associe à toute formule γ de $\mathrm{PSQ}_n(\vec{x})$ un ensemble $\gamma_{|\mathcal{G}|}$ de k-uplets de sommets de $\mathcal{G}$ qui valident γ. L'hypergraphe $\mathrm{Def}_\Delta(\mathcal{G})$ est tel que

– son ensemble de sommets est $\{v \in V \mid v \in \delta_{|\mathcal{G}|}\}$,

– son ensemble d'hyperarcs est

$$\left\{(l, v_1, \dots, v_{\rho(l)}) \mid l \in L, v_1, \dots, v_{\rho(l)} \in V \text{ et } (v_1, \dots, v_{\rho(l)}) \in \eta_{l|\mathcal{G}|}\right\},$$

– son application source associe la σ_i-ième source de $\mathcal{G}$ à tout $i \in [n]$.

Constantes. On considère deux types de constantes, les constantes 0_n et 1_n. Pour tout $n \in \mathbb{N}$, la constante 0_n dénote l'hypergraphe discret de sorte n dont les sommets sont les éléments de $[n]$, l'ensemble d'hyperarcs est vide et la i-ième source est i pour tout i dans $[n]$. Pour tout $n \in \mathbb{N}$, la constante 1_n dénote l'hypergraphe de sorte n dont le seul sommet est 1, l'ensemble d'hyperarcs est $\{(l, 1 \dots 1) \mid l \in L\}$ et la i-ième source est 1 pour tout i dans $[n]$.

Nous présentons maintenant les notions de *signature*, d'*algèbre* et de *système d'équations régulières*. Elles interviendront lors de la définition des graphes VR-équationnels. Une *signature* Σ est un quadruplet $(\mathcal{S}, \mathcal{F}, \alpha, \rho)$ où $\mathcal{S}$ est un ensemble de *sortes*, $\mathcal{F}$ est un ensemble d'*opérations*, $\alpha : \mathcal{F} \to \mathcal{S}^*$ est l'application *arguments* et $\rho : \mathcal{F} \to \mathcal{S}$ est l'application *résultat*. Ces deux applications définissent le *type* $\alpha(f)\rho(f)$ de tout $f \in \mathcal{F}$.

Une Σ-*algèbre* A est la donnée d'un ensemble A_s pour tout $s \in \mathcal{S}$ et d'une application $f_A : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ pour toute opération f de type $s_1 \dots s_n s$. Soit $k \in \mathbb{N}$, $(x_i)_{i \in [k]}$ une suite de variables sans répétition et $\rho' : \{x_1, \dots, x_k\} \to \mathcal{S}$ une application.

Un *système d'équations régulières* E sur la signature Σ est une suite de la forme $x_1 = t_1, \dots, x_k = t_k$ telle que les t_i sont de la forme $f(x_{j_1}, \dots, x_{j_m})$ où $f \in \mathcal{F}$ a pour type $\rho'(x_{j_1}) \dots \rho'(x_{j_m})\rho'(x_i)$. Un tel système peut être interprété dans n'importe quelle Σ-algèbre A et induit une application $E_A : A_{\rho'(x_1)} \times \cdots \times A_{\rho'(x_k)} \to A_{\rho'(x_1)} \times \cdots \times A_{\rho'(x_k)}$ définie naturellement.

On considère la signature Σ dont l'ensemble des sortes est $\mathbb{N}$ et l'ensemble $\mathcal{F}$ des opérations est l'ensemble VR que l'on a décrit ci-dessus (les applications arguments et résultat étant alors définies de façon naturelle). On considère également l'algèbre $\mathscr{H}$ des hypergraphes construite sur Σ. Tout système $E = \{x_1 = t_1, \dots, x_k = t_k\}$ d'équations régulières sur Σ est tel que l'ensemble des points fixes de $E_{\mathscr{H}}$ forme une catégorie dont l'objet initial est la colimite du diagramme suivant [AK79, Bau91] :

$$\vec{0} \to E_{\mathscr{H}}(\vec{0}) \to \dots \to E^i_{\mathscr{H}}(\vec{0}) \to \dots$$

Le symbole $\vec{0}$ représente ici le k-uplet $(0_{\rho(x_1)}, \dots, 0_{\rho(x_k)})$ formé d'hypergraphes discrets et $E^i_{\mathscr{H}}(\vec{0})$ est l'hypergraphe obtenu en appliquant i fois le foncteur $E_{\mathscr{H}}$ à $\vec{0}$. L'objet initial de cette catégorie des points fixes existe toujours et est appelé *plus petite solution* de E.

Enfin, un *hypergraphe est VR-équationnel* lorsqu'il est une composante (*i.e.* la valeur d'un x_i) de la plus petite solution d'un système d'équations régulières faisant intervenir des opérations VR. Klaus Barthelmann a montré dans [Bar97] que la classe des hypergraphes VR-équationnels à une seule source et étiquetés par des symboles de rang égal à deux est isomorphe à celle des graphes préfixe-reconnaissables.

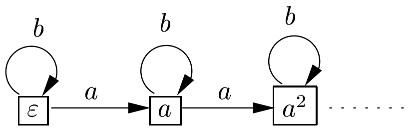
2.2 Décidabilité de la théorie monadique du second ordre

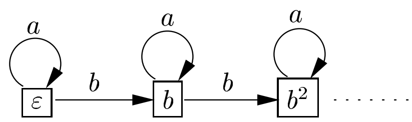
Étant donné un graphe $\mathcal{G}$ préfixe-reconnaissable, la théorie monadique du second ordre de $|\mathcal{G}|_1$ est décidable. En effet, Michael Rabin a montré dans [Rab69] que pour l'arbre binaire complet Λ, la théorie monadique du second ordre de $|\Lambda|_1$ est décidable. De plus, Didier Caucal a montré dans [Cau96] que les opérations « substitution rationnelle inverse » et « restriction rationnelle » préservent la décidabilité de la théorie monadique du second ordre associée à une structure de la forme $|\ldots|_1$. Rappelons que tout graphe préfixe-reconnaissable peut être obtenu, à isomorphisme près, à partir de l'arbre binaire complet par une substitution rationnelle inverse suivie d'une restriction rationnelle.

De son côté, Klaus Barthelmann a montré dans [Bar97] que tout graphe VR-équationnel est l'image d'un arbre par une application (appelée *évaluation*) définissable en logique monadique du second ordre. Il arrive donc au même résultat de décidabilité que Didier Caucal en utilisant également le théorème de Rabin.

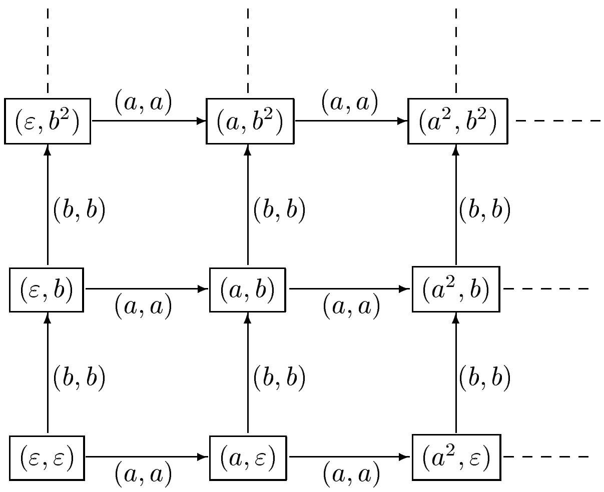
2.3 Produit synchronisé

La classe des graphes préfixe-reconnaissables n'est pas fermée par produit synchronisé. Cela est clairement montré par l'exemple suivant. On considère les deux graphes représentés ci-dessous et la contrainte de synchronisation $\mathscr{C} = \{(a, a), (b, b)\}$:





Ces deux graphes sont préfixe-reconnaissables car ce sont les graphes de transition de deux machines à pile (voir le paragraphe 4). Leur produit synchronisé selon $\mathscr{C}$ est le graphe :



c'est à dire la grille à deux dimensions. On sait (voir par exemple [Tho90]) que la théorie monadique du second ordre d'une telle grille n'est pas décidable, donc ce graphe n'est pas préfixe-reconnaissable.

3 Les graphes réguliers

Les éléments de cette classe peuvent être présentés au moyen de systèmes d'équations régulières faisant intervenir des opérations de remplacement d'hyperarcs. On peut également considérer ces systèmes comme étant des grammaires HR déterministes.

Nous donnons ci-après les définitions précises de ces deux présentations. Nous nous intéressons également à deux caractérisations connues de la classe entière des graphes réguliers dans celle des graphes préfixe-reconnaissables. L'une d'elle, très récente, est due à Didier Caucal et Teodor Knapik.

3.1 Les présentations des graphes réguliers

Grammaires déterministes de graphes

La définition qui suit fait intervenir la notion d'hypergraphe que nous avons introduite au paragraphe 2.1 de ce chapitre (lorsque nous avons présenté les opérations VR).

On considère un ensemble V de sommets et un ensemble fini L d'étiquettes muni d'une application rang $\rho : L \to \mathbb{N} \setminus \{0\}$.

Une *grammaire de graphe* sur V et L est un ensemble fini de règles de la forme $e \to H$ où e est un hyperarc et H un hypergraphe formés sur L et V et H contient les sommets de e. Les étiquettes des membres gauches des règles sont appelées *non-terminaux*. Les autres étiquettes sont appelées *terminaux* et ont un rang égal à deux. Une telle grammaire est déterministe lorsqu'il n'y a qu'une seule règle pour chaque non-terminal.

Réécrire un hypergraphe $M = (V_M, E_M, \mathrm{src}_M)$, c'est remplacer un hyperarc $(l, v_1, \ldots, v_p)$ de M étiqueté par un non-terminal par le membre droit d'une règle de la forme $(l, x_1, \ldots, x_p) \to H$. Les sommets x_i de H indiquent comment effectuer ce remplacement : en appelant V_H l'ensemble des sommets de H (on suppose que $V_M \cap V_H = \varnothing$) et E_H son ensemble d'hyperarcs, l'hypergraphe obtenu a pour ensemble de sommets $V_M \cup \{g(t) \mid t \in V_H\}$ et pour ensemble d'hyperarcs

$$\big(E_M \setminus \{(l, v_1, \ldots, v_p)\}\big) \;\cup\; \big\{(k, g(t_1), \ldots, g(t_q)) \mid (k, t_1, \ldots, t_q) \in E_H\big\}$$

où g est la fonction :

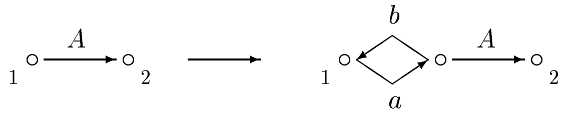
$$\begin{array}{rccc} g : & V_H & \longrightarrow & V_M \cup V_H \\ & x_i \in \{x_1, \ldots, x_p\} & \longmapsto & v_i \\ & t \notin \{x_1, \ldots, x_p\} & \longmapsto & t \end{array}$$

Une *réécriture parallèle* de M consiste à réécrire tous les hyperarcs de M étiquetés par un non-terminal.

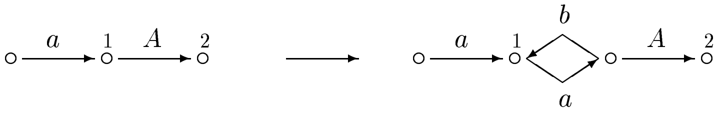
Pour tout hypergraphe N, on note $[E_N]$ l'ensemble des hyperarcs de N étiquetés par un terminal. On dit qu'un graphe G est *engendré* par une grammaire déterministe de graphe à partir d'un hypergraphe M si G est isomorphe au graphe dont les arcs sont $\bigcup_{n \in \mathbb{N}} [E_{M_n}]$ (où $M_0 = M$ et M_{n+1} est obtenu par réécriture parallèle de M_n) et les sommets sont tous ceux qui interviennent dans ces arcs.

Un graphe est *régulier* lorsqu'il est engendré par une grammaire déterministe de graphe à partir d'un hypergraphe fini.

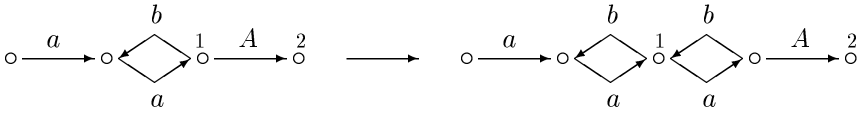
Exemple 3.4 *On considère la grammaire déterministe de graphe constituée de la règle :*



On a alors la réécriture suivante :

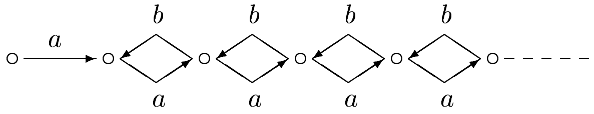


puis



etc.

qui engendre finalement le graphe infini :



Systèmes d'équations avec opérations HR

Les opérations HR sont les suivantes. Elles s'appliquent toutes à des hypergraphes et sont également des opérations VR.

Somme disjointe. Soit $\mathcal{G}$ et $\mathcal{G}'$ deux hypergraphes de sortes respectives n et n' et dont les ensembles de sommets (resp. arcs) sont disjoints. La somme disjointe de $\mathcal{G}$ et $\mathcal{G}'$, notée $\mathcal{G} \oplus \mathcal{G}'$, est un hypergraphe de sorte $n + n'$ et qui est défini comme suit :

- l'ensemble des sommets est l'union des ensembles de sommets,
- l'ensemble des arcs est l'union des ensembles d'arcs,
- l'application source associe, à tout i compris entre 1 et n, la i^{e} source de $\mathcal{G}$ et, à tout i compris entre $n+1$ et $n+n'$, la $(i-n)^{\text{e}}$ source de $\mathcal{G}'$.

Cette opération n'est pas commutative, ce qui est dû à la définition de l'application source.

Fusion de sources. Soit $n \in \mathbb{N}$ et δ une relation d'équivalence sur $[n]$. Soit $\mathcal{G}$ un hypergraphe de sorte n. Alors, $\theta_\delta(\mathcal{G})$ est l'hypergraphe de sorte n obtenu à partir de $\mathcal{G}$ en fusionnant sa i^{e} source à sa j^{e} source pour tout (i, j) dans δ.

Redéfinition de sources. Soit n et p deux entiers naturels. Soit $\alpha : [p] \to [n]$ une application et $\mathcal{G}$ un hypergraphe de sorte n. Alors, $\sigma_\alpha(\mathcal{G})$ est l'hypergraphe de sorte p dont l'ensemble des sommets et des arcs sont ceux de $\mathcal{G}$ et dont l'application source est $\mathrm{src} \circ \alpha$, où src est l'application source de $\mathcal{G}$.

Voici à présent la définition des hypergraphes HR-équationnels. Les notions utilisées ci-après sont définies au paragraphe 2.1 de ce chapitre (là où sont présentés les hypergraphes VR-équationnels). On considère la signature Σ dont l'ensemble des sortes est $\mathbb{N}$ et l'ensemble des opérations est l'ensemble des opérations HR que l'on a décrites ci-dessus (les applications arguments et résultat étant alors définies de façon naturelle). Un *hypergraphe est HR-équationnel* lorsqu'il est une composante (*i.e.* la valeur d'un x_i) de la plus petite solution (qui existe toujours) d'un système d'équations régulières sur Σ.

Les hypergraphes HR-équationnels à une seule source et étiquetés par des symboles de rang égal à deux sont les mêmes que ceux engendrés par les grammaires déterministes de graphe.

3.2 Quelques caractéristiques

Les résultats que nous rapportons ci-après ont été introduits par Didier Caucal. Le premier fournit une caractérisation des graphes réguliers en termes des sous-graphes qui les constituent.

Proposition 3.5 ([MS85, Cau95]) *Un graphe régulier n'a qu'un nombre fini de composantes connexes non-isomorphes par décomposition par distance à partir d'un sommet quelconque.*

Dans cette proposition, la notion de connexité ne tient pas compte de l'orientation et de l'étiquette des arcs. Un graphe $\mathcal{G}$ étiqueté par Σ est connexe si et seulement si pour tous sommets v et v' il existe dans $\mathcal{G}$ un chemin de la forme $v_0 \underset{\mathcal{G}}{\overset{a_1}{\longleftrightarrow}} v_1 \underset{\mathcal{G}}{\overset{a_2}{\longleftrightarrow}} \ldots \underset{\mathcal{G}}{\overset{a_n}{\longleftrightarrow}} v_n$ avec $v_0 = v$, $v_n = v'$ et pour tout $a \in \Sigma$, $\underset{\mathcal{G}}{\overset{a}{\longleftrightarrow}} \;=\; \underset{\mathcal{G}}{\overset{a}{\longrightarrow}} \cup \big(\underset{\mathcal{G}}{\overset{a}{\longrightarrow}}\big)^{-1}$. De plus, la décomposition par distance d'un graphe $\mathcal{G} = \big(V, E\big)$ à partir d'un sommet r est définie comme suit. Pour commencer, étant donnés deux sommets v et v', la distance de v à v' dans $\mathcal{G}$ est

$$d_{\mathcal{G}}(v, v') = \min\big(\{n \mid v \underset{\mathcal{G}}{\longleftrightarrow}^{n} v'\} \;\cup\; \{\infty\}\big)$$

où $\underset{\mathcal{G}}{\longleftrightarrow}$ est la relation $\bigcup\limits_{a \in \Sigma} \underset{\mathcal{G}}{\overset{a}{\longleftrightarrow}}$ et $\underset{\mathcal{G}}{\longleftrightarrow}^{n}$ est la composée n fois de $\underset{\mathcal{G}}{\longleftrightarrow}$. Ensuite, on considère, pour tout entier naturel n, un graphe $\mathcal{G}_{n,r}$ isomorphe à

$$\Big(V \smallsetminus \big\{v \mid d_{\mathcal{G}}(r, v) \leq n\big\},\; E \smallsetminus \big\{v \overset{a}{\to} v', v' \overset{a}{\to} v \mid d_{\mathcal{G}}(r, v) \leq n\big\}\Big)$$

et d'ensemble de sommets $V_{n,r}$ disjoint de V. Enfin, la décomposition par distance de $\mathcal{G}$ à partir de r est le graphe obtenu en regroupant $\mathcal{G}$ et les $\mathcal{G}_{n,r}$, *i.e.* est le graphe

$$\text{décomposition}(\mathcal{G}, r) = \Big(V \cup \bigcup_{n \in \mathbb{N}} V_{n,r},\; E \cup \bigcup_{n \in \mathbb{N}} E_{n,r}\Big)$$

en supposant que pour tout n et tout n', les ensembles $V_{n,r}$ et $V_{n',r}$ sont disjoints et, pour tout n, $E_{n,r}$ est l'ensemble des arcs de $\mathcal{G}_{n,r}$. Le résultat de la proposition signifie donc que le quotient par isomorphisme de l'ensemble des composantes connexes de décomposition$(\mathcal{G}, r)$ est fini.

Le second résultat que nous mentionnons caractérise les traces des graphes réguliers, c'est-à-dire les langages formés des étiquettes des chemins allant d'un ensemble fini de sommets à un autre.

Proposition 3.6 *Les traces des graphes réguliers sont les langages algébriques.*

Notons que cette proposition reste vraie pour les graphes préfixe-reconnaissables.

3.3 Caractérisation dans la classe des graphes préfixe-reconnaissables

Didier Caucal a montré, dans [Cau96], que la classe des graphes réguliers est strictement incluse dans celle des graphes préfixe-reconnaissables. Ceci prouve en particulier que la théorie monadique du second ordre de $|\mathcal{G}|_1$, pour tout graphe régulier $\mathcal{G}$, est décidable. Toutefois, Bruno Courcelle avait déjà montré dans [Cou89] que la théorie monadique du second ordre avec comptage modulaire de $|\mathcal{G}|_2$ est décidable lorsque $\mathcal{G}$ est équationnel (par rapport aux opérations HR). Dans sa démonstration, il fait remarquer que tout graphe équationnel est l'image d'un arbre par une application qui est définissable en logique monadique du second ordre.

Peu de caractérisations de la classe entière des graphes réguliers dans celle des graphes préfixe-reconnaissables sont connues actuellement. En voici quelques-unes décrites brièvement. Klaus Barthelmann a montré dans [Bar98] qu'un graphe préfixe-reconnaissable est régulier si et seulement si l'ensemble de ses sous-graphes bipartis complets est fini à isomorphisme près. D'autre part, Didier Caucal et Teodor Knapik donnent dans [CK99a] plusieurs caractérisations internes dont celles qui suivent.

Soit $\mathcal{G}$ un graphe préfixe-reconnaissable étiqueté par Σ et tel que, pour tout $a \in \Sigma$,

$$\underset{\mathcal{G}}{\overset{a}{\rightarrow}} = \bigcup_{i \in I_a} (U_{a,i} \times V_{a,i}) W_{a,i} \ .$$

On a alors les deux caractérisations suivantes.

- $\mathcal{G}$ est régulier si et seulement si au moins l'une des deux conditions suivantes est vérifiée pour tout $a \in \Sigma$ et tout $i \in I_a$.
  - $|U_{a,i}| < \infty$ et il n'y a pas de mots x, y et z tels que $x, y \neq \varepsilon$, $xy^* \subseteq V_i$ et $y^*z \subseteq W_{a,i}$,
  - $|U_{a,i}| < \infty$ et il n'y a pas de mots x, y et z tels que $x, y \neq \varepsilon$, $xy^* \subseteq U_i$ et $y^*z \subseteq W_{a,i}$.
- $\mathcal{G}$ est régulier si et seulement si son nombre de degrés est fini. Voici comment est défini ce nombre. Pour tout mot w de Σ^* et toute lettre a de Σ, on appelle
  - degré sortant de w dans $\underset{\mathcal{G}}{\overset{a}{\rightarrow}}$ le nombre $d^+(w, \underset{\mathcal{G}}{\overset{a}{\rightarrow}}) = \big|\{w' \mid w \underset{\mathcal{G}}{\overset{a}{\rightarrow}} w'\}\big|$,
  - degré entrant de w dans $\underset{\mathcal{G}}{\overset{a}{\rightarrow}}$ le nombre $d^-(w, \underset{\mathcal{G}}{\overset{a}{\rightarrow}}) = \big|\{w' \mid w' \underset{\mathcal{G}}{\overset{a}{\rightarrow}} w\}\big|$,
  - degré de w dans $\underset{\mathcal{G}}{\overset{a}{\rightarrow}}$ le nombre $d(w, \underset{\mathcal{G}}{\overset{a}{\rightarrow}}) = d^+(w, \underset{\mathcal{G}}{\overset{a}{\rightarrow}}) + d^-(w, \underset{\mathcal{G}}{\overset{a}{\rightarrow}})$.

  Le nombre de degrés de $\underset{\mathcal{G}}{\overset{a}{\rightarrow}}$ est $d(\underset{\mathcal{G}}{\overset{a}{\rightarrow}}) = \big|\{d(w, \underset{\mathcal{G}}{\overset{a}{\rightarrow}}) \mid w \in \Sigma^*\}\big|$. Enfin, le nombre de degrés de $\mathcal{G}$ est $\sum_{a \in \Sigma} d(\underset{\mathcal{G}}{\overset{a}{\rightarrow}})$.

3.4 Produit synchronisé

La classe des graphes réguliers n'est pas fermée par produit synchronisé. L'exemple du paragraphe 2.3 de ce chapitre le montre car les graphes dont on fait le produit sont aussi réguliers (ce sont les graphes de transition de deux machines à pile, voir le paragraphe 4).

4 Les graphes des machines à pile

Historiquement, c'est la première des trois classes de graphes exposées ici à avoir été présentée. Ses éléments ont été introduits par David Muller et Paul Schupp dans [MS85] sous le nom de *graphes context-free*. Ces graphes constituent une généralisation des graphes de Cayley des groupes context-free.

4.1 Graphe de transitions d'une machine à pile

Une machine à pile $\mathcal{P}$ est un quintuplet $(Q, \Sigma, \Gamma, \delta, q_0)$ où Q est l'ensemble fini des *états*, Σ est l'*alphabet d'entrée*, Γ est l'*alphabet de pile*, δ est l'ensemble fini des *règles de transition* vérifiant $\delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \times \Gamma^* \times Q$ et q_0 est l'*état initial*. On suppose qu'initialement la pile d'une telle machine est vide.

Une *configuration interne* de $\mathcal{P}$ est un élément de $Q \times \Gamma^*$. Étant donnée une configuration interne ι, on associe à $\mathcal{P}$ le graphe simple, orienté et étiqueté $\mathcal{G}(\mathcal{P}, \iota)$ défini comme suit. Les sommets du graphe sont toutes les configurations internes accessibles à partir de ι qui est la racine distinguée du graphe. Il y a un arc étiqueté par $a \in \Sigma \cup \{\varepsilon\}$ de (q_1, h_1) à (q_2, h_2) s'il existe une lettre $X \in \Gamma$ et deux mots g_1 et g_2 dans Γ^* tels que $h_1 = g_1X$, $h_2 = g_1g_2$ et $(q_1, a, X, g_2, q_2) \in \delta$.

4.2 Caractérisation dans la classe des graphes réguliers

Les graphes de transitions des machines à pile forment un ensemble complet de représentants des graphes réguliers de degré fini ayant une racine distinguée. Ceci est démontré dans [Cau92] où l'auteur fournit une technique effective de transformation de toute grammaire déterministe de graphe engendrant un graphe de degré fini et ayant une racine distinguée en une machine à pile et une configuration interne (la racine du graphe). Il fournit également une technique de transformation dans le sens inverse.

4.3 Décidabilité de la théorie monadique du second ordre

Muller et Schupp ont montré dans [MS85] qu'étant données une machine à pile $\mathcal{P}$ et une configuration interne ι, la théorie monadique du second ordre de $|\mathcal{G}(\mathcal{P}, \iota)|_1$ est décidable. Ce résultat a plus tard été étendu par Courcelle et Bauderon qui ont montré que tout graphe de machine à pile est équationnel [Bau92] et que pour tout graphe équationnel $\mathcal{G}$, la théorie monadique du second ordre avec comptage modulaire de $|\mathcal{G}|_2$ est décidable [Cou89].

4.4 Produit synchronisé

La classe des graphes de transitions des machines à pile n'est pas fermée par produit synchronisé. Il suffit de considérer les machines

$$\mathcal{P}_1 = \big(\{q_\varepsilon\}, \{a, b\}, \{a, b, z\}, \{(q_\varepsilon, a, z, za, q_\varepsilon), (q_\varepsilon, a, a, aa, q_\varepsilon), (q_\varepsilon, b, z, z, q_\varepsilon), (q_\varepsilon, b, a, a, q_\varepsilon)\}\big)$$

et

$$\mathcal{P}_2 = \big(\{q_\varepsilon\}, \{a, b\}, \{a, b, z\}, \{(q_\varepsilon, a, z, z, q_\varepsilon), (q_\varepsilon, a, b, b, q_\varepsilon), (q_\varepsilon, b, z, zb, q_\varepsilon), (q_\varepsilon, b, b, bb, q_\varepsilon)\}\big)$$

dont les graphes $\mathcal{G}(\mathcal{P}_1, (q_\varepsilon, z))$ et $\mathcal{G}(\mathcal{P}_2, (q_\varepsilon, z))$ sont isomorphes à ceux de l'exemple présenté au paragraphe 2.3 de ce chapitre.

Chapitre 4

Graphes des spécifications de Thue

Sommaire

Les spécifications de Thue ont été introduites par Teodor Knapik [Kna96] dans le but de fournir une nouvelle approche de spécification de systèmes informatiques et de vérification de propriétés dynamiques grâce à des techniques de la théorie des langages et de la réécriture de mots.

Une spécification de Thue est la donnée, entre autres, d'un système de réécriture de mots et d'un sous-ensemble rationnel de l'ensemble des mots irréductibles par le système. Les graphes associés à ces spécifications sont étudiés dans ce chapitre. Nous présentons les liens qui les unissent à ceux présentés au chapitre précédent. Nous montrons ensuite que la classe des graphes des spécifications de Thue est fermée par produit synchronisé. Ce résultat constitue notre contribution principale à l'étude de ces graphes.

1 Définitions

1.1 Semi-systèmes de Thue

Un *semi-système de Thue* (*système* en abrégé) $\mathcal{S}$ sur un alphabet Σ est un sous-ensemble de $\Sigma^* \times \Sigma^*$. Un couple (l, r) dans $\mathcal{S}$ est appelé *règle (de réécriture)* et est plutôt noté $l \to r$. Le mot l (resp. r) est le membre gauche (resp. droit) de la règle. Dans cette thèse, nous ne considérons que des systèmes ayant un nombre fini de règles.

La *relation de réduction en une étape* induite par $\mathcal{S}$ est la relation binaire $\underset{\mathcal{S}}{\to}$ définie par

$$\underset{\mathcal{S}}{\to} = \left\{ (u,v) \in \Sigma^* \times \Sigma^* \;\middle|\; \exists\, x,y \in \Sigma^*,\ \exists\, l \to r \in \mathcal{S},\ u = xly \text{ et } v = xry \right\} .$$

La *relation de réduction* $\overset{*}{\underset{\mathcal{S}}{\to}}$ induite par $\mathcal{S}$ est la fermeture réflexive et transitive de $\underset{\mathcal{S}}{\to}$.

Un mot u est *réductible* par $\mathcal{S}$ si u appartient à $\mathcal{D}\mathrm{om}(\underset{\mathcal{S}}{\to})$. Sinon, u est *irréductible* par $\mathcal{S}$. L'ensemble de tous les mots irréductibles par $\mathcal{S}$, noté $\mathrm{Irr}(\mathcal{S})$, est rationnel car $\mathcal{D}\mathrm{om}(\mathcal{S})$ est fini. En effet, on vérifie facilement que $\mathrm{Irr}(\mathcal{S}) = \Sigma^* \smallsetminus \Sigma^*.\mathcal{D}\mathrm{om}(\mathcal{S}).\Sigma^*$. Un mot v est une *forme normale* de u si v est irréductible et $u \overset{*}{\underset{\mathcal{S}}{\to}} v$. L'ensemble des formes normales de u est noté $u{\downarrow_{\mathcal{S}}}$.

Dans la suite de ce document, nous considérons les semi-systèmes de Thue particuliers suivants :

Semi-systèmes de Thue linéaires. Le système $\mathcal{S} \subseteq \Sigma^* \times \Sigma^*$ est linéaire s'il existe une fonction linéaire $f : \mathbb{N} \to \mathbb{N}$ vérifiant : $\forall w, w' \in \Sigma^*,\ w \overset{*}{\underset{\mathcal{S}}{\to}} w' \Rightarrow |w'| \leq f\big(|w|\big)$.

Semi-systèmes de Thue longueur-décroissantes. Le système $\mathcal{S} \subseteq \Sigma^* \times \Sigma^*$ est longueur-décroissante s'il vérifie : $\forall w, w' \in \Sigma^*,\ w \overset{*}{\underset{\mathcal{S}}{\to}} w' \Rightarrow |w'| \leq |w|$.

Les semi-systèmes de Thue suivants ont été introduits dans [CK98] et [KC99] (dans les définitions qui suivent, on se donne un système $\mathcal{S} \subseteq \Sigma^* \times \Sigma^*$) :

Semi-systèmes de Thue suffixe-bornés. Soit L un sous-ensemble de $\mathrm{Irr}(\mathcal{S})$. La borne-suffixe de $\mathcal{S}$ sur L, notée $\beta(\mathcal{S}, L)$, est définie par

$$\beta(\mathcal{S}, L) = \max \left\{ |ly| \;\middle|\; w \in L,\ a \in \Sigma,\ x,y \in \Sigma^*,\ l \to r \in \mathcal{S},\ wa \overset{*}{\underset{\mathcal{S}}{\to}} xly \right\} .$$

Lorsque $\beta(\mathcal{S}, L)$ est finie, on dit que $\mathcal{S}$ est suffixe-borné sur L. Si $\mathcal{S}$ est suffixe-borné sur $\mathrm{Irr}(\mathcal{S})$, on dit que $\mathcal{S}$ est uniformément suffixe-borné.

Semi-systèmes de Thue suffixes. La relation de réduction-suffixe en une étape induite par $\mathcal{S}$ est définie par :

$$\underset{\mathcal{S}}{\twoheadrightarrow} = \left\{ (xl, xr) \mid x \in \Sigma^*,\ l \to r \in \mathcal{S} \right\}$$

et la fermeture réflexive et transitive de $\underset{\mathcal{S}}{\twoheadrightarrow}$ est notée $\overset{*}{\underset{\mathcal{S}}{\twoheadrightarrow}}$ et est appelée relation de réduction-suffixe induite par $\mathcal{S}$. On dit que $\mathcal{S}$ est suffixe sur $L \subseteq \mathrm{Irr}(\mathcal{S})$ si et seulement si

$$(L.\Sigma \times \Sigma^*) \cap \overset{*}{\underset{\mathcal{S}}{\twoheadrightarrow}} = (L.\Sigma \times \Sigma^*) \cap \overset{*}{\underset{\mathcal{S}}{\to}} .$$

Lorsque $\mathcal{S}$ est suffixe sur $\mathrm{Irr}(\mathcal{S})$, on dit que $\mathcal{S}$ est uniformément suffixe.

Semi-systèmes de Thue fortement réduction-bornées. Le système $\mathcal{S}$ est fortement réduction-bornée s'il existe un entier naturel k tel que pour tout $w \in \mathrm{Irr}(\mathcal{S})$ et tout $a \in \Sigma$, la longueur de toute réduction de wa par les règles de $\mathcal{S}$ ne dépasse pas k. On dit alors que le nombre k est une borne de $\mathcal{S}$. Lorsque $k = 1$, on dit que $\mathcal{S}$ est à réduction bornée par l'unité.

Notons que tout système monadique (*i.e.* dont toutes les règles sont de la forme $w \to a$ avec $|a| \leq 1$ et $|w| > |a|$) est également suffixe-borné. Il en va de même pour tout système $\mathcal{S}$ fortement réduction-bornée car $\mathcal{S}$ est suffixe-borné sur n'importe quel sous-ensemble de $\mathrm{Irr}(\mathcal{S})$. En effet, on a $\beta(\mathcal{S}, \mathrm{Irr}(\mathcal{S})) \leq k.\max_{l \in \mathfrak{Dom}(\mathcal{S})}\{\ |l|\ \}$ où k est une borne de $\mathcal{S}$. Enfin, tout système $\mathcal{S}$ suffixe sur L est aussi suffixe-borné sur L car $\beta(\mathcal{S}, L) = \max_{l \in \mathfrak{Suff}(L.\Sigma) \cap \mathfrak{Dom}(\mathcal{S})}\{\ |l|\ \}$ est finie.

Les résultats de décidabilité suivants sont démontrés dans [KC99].

Théorème 4.1 *Le problème suivant est indécidable :*

> ***Instance :*** *Un semi-système de Thue $\mathcal{S}$.*
> ***Question :*** *$\mathcal{S}$ est-il uniformément suffixe-borné ?*

Théorème 4.2 *Il existe un algorithme qui résout le problème suivant :*

> ***Instance :*** *Un semi-système de Thue $\mathcal{S}$ et un sous-ensemble rationnel R de $Irr(\mathcal{S})$.*
> ***Question :*** *$\mathcal{S}$ est-il suffixe sur R ?*

Pour ce qui est des semi-systèmes de Thue fortement réduction-bornées, les résultats suivants sont vérifiés [CK98].

Théorème 4.3 *Le problème suivant est indécidable :*

> ***Instance :*** *Un semi-système de Thue $\mathcal{S}$.*
> ***Question :*** *$\mathcal{S}$ est-il fortement réduction-bornée ?*

Théorème 4.4 *Il existe un algorithme qui résout le problème suivant :*

> ***Instance :*** *Un semi-système de Thue $\mathcal{S}$.*
> ***Question :*** *$\mathcal{S}$ est-il à réduction bornée par l'unité ?*

1.2 Spécifications de Thue

Une spécification de Thue est un quintuplet $\mathcal{T} = (\Omega, \Delta, \mathcal{S}, R, u)$ où Ω et Δ sont des alphabets appelés respectivement *alphabet d'états* et *alphabet d'événements*, $\mathcal{S}$ est un semi-système de Thue sur $\Omega \cup \Delta$, $R \subseteq \Omega^*$ est un sous-ensemble rationnel de $\mathrm{Irr}(\mathcal{S})$ et $u \in R$.

Cette définition implique que $\mathcal{S}$ ne contient aucune règle dont le membre gauche est le mot vide (sinon, $\mathrm{Irr}(\mathcal{S}) = \varnothing$, $R = \varnothing$ et il ne peut exister de u satisfaisant la définition).

La spécification de Thue $\mathcal{T}$ est *linéaire* (resp. *longueur-décroissante*, resp. *fortement réduction-bornée*) si son semi-système de Thue $\mathcal{S}$ l'est. On dit de plus que $\mathcal{T}$ est *suffixe-bornée* (resp. *suffixe*) si $\mathcal{S}$ est suffixe-borné (resp. suffixe) sur R.

1.3 Graphe d'une spécification de Thue

Soit $\mathcal{T} = (\Omega, \Delta, \mathcal{S}, R, u)$ une spécification de Thue. Soit $\mathcal{G}$ le graphe défini comme suit. Les sommets de $\mathcal{G}$ sont les mots de R, les arcs de $\mathcal{G}$ sont étiquetés par les lettres de Δ et il y a un arc étiqueté par a de v à w si w est une forme normale de va. Le graphe de $\mathcal{T}$, noté $\mathcal{G}(\mathcal{T})$, est défini comme étant le couple $(\mathcal{G}', u)$ [1] où $\mathcal{G}'$ est le sous-graphe maximal de $\mathcal{G}$ dont u est une racine.

Nous adopterons, dans la suite de ce document, les notations suivantes : $\mathcal{G}$[TS] désigne la classe des graphes des spécifications de Thue, $\mathcal{G}$[Lin-TS] celle des graphes des spécifications de Thue linéaires et $\mathcal{G}$[Len-TS] celle des graphes des spécifications de Thue longueur-décroissantes.

Notons que la classe des graphes des spécifications suffixe-bornées et celle des graphes des spécifications suffixes sont liées de la façon suivante.

Théorème 4.5 ([KC99]) *Étant donnée une spécification de Thue $\mathcal{T}$ suffixe-bornée, on peut construire une spécification $\mathcal{T}'$ suffixe telle que les graphes $\mathcal{G}(\mathcal{T})$ et $\mathcal{G}(\mathcal{T}')$ sont isomorphes.*

Ces deux classes de graphes sont donc isomorphes mais les résultats de décidabilité associés aux systèmes suffixe-bornés et aux systèmes suffixes ne sont pas les mêmes, comme nous l'avons indiqué au paragraphe 1.1. Néanmoins, la construction d'une spécification de Thue suffixe à partir d'une spécification de Thue suffixe-bornée n'est effective que si $\beta(\mathcal{S}, R)$ peut être calculé, ce qui n'est pas le cas en général.

2 Liens avec les autres classes de graphes

2.1 Quelques liens avec les graphes des machines à pile et avec les graphes préfixe-reconnaissables

Les résultats présentés ici sont dus à Hugues Calbrix et Teodor Knapik [CK98] et à Didier Caucal et Teodor Knapik [CK99b].

La première de ces deux références fournit une caractérisation des graphes des machines à pile au moyen de systèmes de réécriture de mots. Les auteurs prouvent les deux théorèmes suivants de façon constructive, ce qui permet de passer d'une présentation de ces graphes sous la forme d'un automate à pile à une présentation sous la forme d'une spécification de Thue et vice versa.

Théorème 4.6 *Étant donnés une machine à pile temps-réel (*i.e.* sans transition lisant ε sur la bande d'entrée) $\mathcal{P}$ et une configuration interne ι de $\mathcal{P}$, on peut construire une spécification $\mathcal{T}$ à réduction bornée par l'unité telle que les graphes $\mathcal{G}(\mathcal{P}, \iota)$ et $\mathcal{G}(\mathcal{T})$ sont isomorphes.*

Inversement, étant donnée une spécification de Thue $\mathcal{T}$ à réduction bornée par l'unité, on peut construire une machine à pile temps-réel $\mathcal{P}$ et une configuration interne ι tels que les graphes $\mathcal{G}(\mathcal{T})$ et $\mathcal{G}(\mathcal{P}, \iota)$ sont isomorphes.

Théorème 4.7 *Étant donnée une spécification de Thue $\mathcal{T}$ suffixe-bornée, on peut construire une machine à pile $\mathcal{P}$ et une configuration interne ι tels que les graphes $\mathcal{G}(\mathcal{T})$ et $\mathcal{G}(\mathcal{P}, \iota)$ sont ε-équivalents.*

1. Nous rappelons au lecteur que cette notation, définie au chapitre 2, signifie que $\mathcal{G}'$ est un graphe dont r est une racine que l'on distingue.

Les résultats présentés dans [CK99b] établissent des liens entre la classe des graphes des spécifications de Thue et celle des graphes préfixe-reconnaissables. Parmi ces liens, nous avons relevé celui-ci :

Théorème 4.8 *La classe des graphes préfixe-reconnaissables avec une racine distinguée est la même que celle des restrictions rationnelles sur les sommets des graphes des spécifications de Thue suffixe-bornées.*

2.2 Un lien avec les graphes rationnels

On considère un graphe rationnel $\mathcal{G}$ dont les sommets sont des éléments de Z^* (Z étant un alphabet fini quelconque) et qui a pour racine distinguée r. On suppose que $\mathcal{G}$ est étiqueté par les éléments d'un alphabet fini, qu'on appelera Σ, qui ne contient pas le mot vide et qui est disjoint de Z.

Pour chaque $a \in \Sigma$, on se donne un transducteur $\mathcal{K}_a$ qui reconnaît $\underset{\mathcal{G}}{\xrightarrow{a}}$. On suppose que l'alphabet d'entrée et l'alphabet de sortie de $\mathcal{K}_a$ sont égaux à Z et on appelle Q_a son ensemble fini d'états, $\delta_a \subseteq Q_a \times Z^* \times Z^* \times Q_a$ son ensemble fini de transitions, q_0^a son état initial et F_a son ensemble d'états acceptants. On suppose enfin que $Q_a \cap Z = \varnothing$ et pour tout $b \in \Sigma$, $b \neq a \Rightarrow Q_a \cap Q_b = \varnothing$.

On définit alors le transducteur $\mathcal{K}$ de la façon suivante :

- son alphabet d'entrée et son alphabet de sortie sont égaux à Z,
- son ensemble d'état est $\{q_0\} \cup \bigcup_{a\in\Sigma} Q_a$ où q_0 est un nouvel état n'appartenant pas à $Z \cup \bigcup_{a\in\Sigma} Q_a$,
- son ensemble de transitions est $\big\{(q_0, \varepsilon, \varepsilon, q_0^a)\big\} \cup \bigcup_{a\in\Sigma} \delta_a$
- son état initial est q_0 et
- son ensemble d'états acceptants est $\bigcup_{a\in\Sigma} F_a$.

Le transducteur $\mathcal{K}$ reconnaît $\bigcup_{a\in\Sigma} \underset{\mathcal{G}}{\xrightarrow{a}}$. À partir de $\mathcal{K}$, on définit la spécification de Thue

$$\mathcal{T} = \big(Q \cup Z \cup \{£\}, \Sigma, \mathcal{S}, R, u\big) \text{ telle que}$$

- $£$ est un nouveau symbole n'appartenant pas à $Q \cup \Sigma \cup Z$,
- le système $\mathcal{S}$ est

$$\begin{aligned}\mathcal{S} = \quad & \big\{£a \to a£ \mid a \in \Sigma\big\} \;\cup\; \big\{za \to az \mid a \in \Sigma,\ z \in Z\big\} \cup \\ & \big\{q_0 a \to q_0^a \mid a \in \Sigma\big\} \cup \\ & \big\{qw \to w'q' \mid (q, w, w', q') \in \textstyle\bigcup_{a\in\Sigma} \delta_a\big\} \cup \\ & \big\{f£ \to q_0£ \mid f \in \textstyle\bigcup_{a\in\Sigma} F_a\big\} \cup \\ & \big\{zq_0 \to q_0 z \mid z \in Z\big\},\end{aligned}$$

- $R = q_0 \,.\, \mathcal{D}\mathrm{om}(\mathbf{R}) \,.\, \pounds$ où $\mathbf{R}$ est la relation reconnue par $\mathcal{K}$ (*i.e.* $\mathbf{R} = \bigcup_{a\in\Sigma} \xrightarrow[\mathcal{G}]{a}$) [2] et
- $u = q_0 r \pounds$.

On appelle ϕ la bijection de Z^* dans R qui transforme tout élément w en l'élément $q_0 w \pounds$. Il est clair que, d'une part, $\Phi(r) = u$ et, d'autre part, $w \xrightarrow[\mathcal{G}]{a} w'$ si et seulement si $q_0 w \pounds \xrightarrow[\mathcal{G}(\mathcal{T})]{a} q_0 w' \pounds$. Les graphes $\mathcal{G}$ et $\mathcal{G}(\mathcal{T})$ sont donc isomorphes. Remarquons que ce résultat est conservé si l'on considère un graphe rationnel $\mathcal{G}$ sans racine distinguée (dans ce cas, la spécification de Thue $\mathcal{T}$ construite n'a pas de « mot initial » u distingué). On peut donc énoncer le résultat suivant.

Théorème 4.9 *La classe des graphes rationnels sans ε-arc et avec (resp. sans) racine distinguée est incluse, à isomorphisme près, dans celle des graphes des spécifications de Thue (resp. des spécifications de Thue sans mot initial u distingué).*

2.3 Conséquences sur la décidabilité de théories

La preuve de l'indécidabilité de la théorie du premier ordre des graphes rationnels présentée par Christophe Morvan dans [Mor00] s'applique tout-à-fait aux graphes rationnels sans ε-arc (cette preuve consiste à réduire le problème de correspondance de Post à celui de la décidabilité d'une formule du premier ordre de la forme $\exists x\ \mathbf{arc}[a](x,x)$ sur un graphe rationnel). On déduit donc du théorème 4.9 que la théorie du premier ordre des graphes des spécifications de Thue est indécidable de façon générale.

Pour ce qui est des graphes des spécifications de Thue suffixe-bornées, il est prouvé dans [KC99] que leur théorie monadique du second ordre est décidable. Les auteurs démontrent cela en utilisant le résultat du théorème 4.7 ainsi qu'une transduction de graphes particulière.

3 Fermeture par produit synchronisé

La classe des graphes des spécifications de Thue est fermée par produit synchronisé. C'est ce que nous allons prouver maintenant grâce à une démonstration constructive : étant données n spécifications et une contrainte de synchronisation, nous construisons une spécification dont le graphe est isomorphe au produit des graphes des spécifications initiales. Nous nous intéressons également à la complexité de cette construction.

Avant de commencer, nous rappelons les notations suivantes. Pour tout alphabet Σ, $\widetilde{\Sigma}$ est l'ensemble $\Sigma \cup \{\varepsilon\}$ et si $\vec{t}$ est un n-uplet, $\pi_i(\vec{t})$ dénote sa i^{e} coordonnée.

3.1 Produit synchronisé de spécifications de Thue

Soit n un entier naturel supérieur ou égal à 2 et $\mathcal{T}_i = (\Omega_i, \Delta_i, \mathcal{S}_i, R_i, u_i)$, $i \in [n]$, des spécifications de Thue. Soit $\mathscr{C} \subseteq \prod_{i\in[n]} \Delta_i$ une contrainte de synchronisation. Le *produit synchronisé*

2. Notons que $\mathcal{D}\mathrm{om}(\mathbf{R})$ est un langage rationnel car il est reconnu par l'automate fini obtenu à partir de $\mathcal{K}$ en transformant toutes les transitions (q,x,y,q') en (q,x,q').

des $\mathcal{T}_i$ selon $\mathscr{C}$ est la spécification de Thue

$$\prod_{i\in[n]}^{\mathscr{C}} \mathcal{T}_i = \big(\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}, \mathscr{C}, \mathcal{S}, R, u\big)$$

dont les composantes $\mathcal{S}$, R et u seront définies dans la suite de cette partie.

Voici en premier lieu quelques définitions dont nous ferons un usage courant.

- On appelle $\mathscr{L}$ le sous-ensemble suivant de $\big[\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\big]^*$:

$$\left\{ w \neq \varepsilon \;\middle|\; \begin{array}{l} \forall i \in [\,|w|-1],\ \forall j \in [n],\ \pi_j\big(w(i)\big) = \varepsilon \Rightarrow \pi_j\big(w(i+1)\big) = \varepsilon \text{ et} \\ \forall i \in [\,|w|\,],\ \exists j \in [n],\ \pi_j\big(w(i)\big) \neq \varepsilon \end{array} \right\} \bigcup \ \{\varepsilon\}\ .$$

- Pour tout $i \in [n]$, on étend l'application π_i à l'ensemble $\big[\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\big]^*$ de la façon suivante : pour tout mot $w \neq \varepsilon$ de cet ensemble, $\pi_i\big(w\big) = \pi_i\big(w(1)\big)\ldots\pi_i\big(w(\,|w|\,)\big)$ et $\pi_i\big(\varepsilon\big) = \varepsilon$.

- On appelle ϕ la bijection définie par :

$$\begin{array}{rccl} \phi : & \prod_{i\in[n]} (\Omega_i \cup \Delta_i)^* & \longrightarrow & \mathscr{L} \\ & \vec{w} & \longmapsto & w' \quad \text{tel que } \forall i \in [n],\ \pi_i\big(w'\big) = \pi_i\big(\vec{w}\big)\ . \end{array}$$

  Cette définition fait usage des applications π_i étendues comme précédemment.

Dans la suite de ce chapitre, nous noterons les vecteurs $(a_1, \ldots, a_n)$ sous la forme verticale $\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$ lorsque nous jugerons que cela confère une meilleure lisibilité et donc une meilleure compréhension des concepts introduits.

Exemple 4.10 *Supposons que $n = 3$ et que pour tout $i \in [n]$, $\Omega_i = \Delta_i = \{a, b, c\}$. Alors, $\mathscr{L}$ contient par exemple les mots*

$$\begin{pmatrix} a \\ a \\ b \end{pmatrix}\begin{pmatrix} b \\ b \\ b \end{pmatrix} \quad et \quad \begin{pmatrix} a \\ b \\ c \end{pmatrix}\begin{pmatrix} b \\ \varepsilon \\ a \end{pmatrix}\begin{pmatrix} \varepsilon \\ \varepsilon \\ a \end{pmatrix}$$

mais ne contient aucun de ces deux mots :

$$\begin{pmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix} \quad et \quad \begin{pmatrix} a \\ b \\ c \end{pmatrix}\begin{pmatrix} b \\ \varepsilon \\ a \end{pmatrix}\begin{pmatrix} b \\ a \\ \varepsilon \end{pmatrix}\ .$$

D'autre part,

$$\phi\begin{pmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix} = \varepsilon \quad et \quad \phi\begin{pmatrix} ab \\ b \\ caa \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}\begin{pmatrix} b \\ \varepsilon \\ a \end{pmatrix}\begin{pmatrix} \varepsilon \\ \varepsilon \\ a \end{pmatrix}\ .$$

Le semi-système de Thue $\mathcal{S}$. Nous définissons ce système comme étant l'union du système $\mathcal{S}_a$ contenant les *règles de réarrangement* et du système $\mathcal{S}_s$ contenant les *règles de simulation.* Les règles de réarrangement sont :

- les $\vec{t}_1\vec{t}_2 \to \vec{u}_1\vec{u}_2$ telles que
  - $\vec{t}_1$, $\vec{t}_2$, $\vec{u}_1$ et $\vec{u}_2$ sont des éléments de $\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}$ et
  - $\vec{t}_1 \neq \vec{\varepsilon}_n$ où $\vec{\varepsilon}_n$ dénote le n-uplet dont toutes les composantes sont égales au mot vide et
  - si $S = \{i \in [n] \mid \pi_i(\vec{t}_1) = \varepsilon \text{ et } \pi_i(\vec{t}_2) \neq \varepsilon\}$, alors $S \neq \varnothing$ et
    - $\forall i \in S$, $\pi_i(\vec{u}_2) = \varepsilon$ et $\pi_i(\vec{u}_1) = \pi_i(\vec{t}_2)$ et
    - $\forall i \notin S$, $\pi_i(\vec{u}_1) = \pi_i(\vec{t}_1)$ et $\pi_i(\vec{u}_2) = \pi_i(\vec{t}_2)$,
- et $\vec{\varepsilon}_n \to \varepsilon$.

Le rôle de ces règles est énoncé dans le lemme suivant.

Lemme 4.11 *Le système de réécriture $\mathcal{S}_a$ est convergent. De plus, pour tout mot w du langage $\left[\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\right]^* \setminus \mathscr{L}$, $w{\downarrow_{\mathcal{S}_a}}$ est le mot w' de $\mathscr{L}$ qui vérifie : pour tout $i \in [n]$, $\pi_i(w') = \pi_i(w)$. Enfin, tout mot de $\mathscr{L}$ est irréductible par $\mathcal{S}_a$.*

Les règles de simulation sont les $w \to w'$ telles que

- $w \in \mathscr{L}$ et $w' \in \mathscr{L}$,
- $\exists i \in [n]$, $\pi_i(w) \to \pi_i(w') \in \mathcal{S}_i$ et $\forall j \in [\,|w|\,]$, $\pi_i(w(j)) \neq \varepsilon$,
- $\forall j \in [n] \setminus \{i\}$, $\pi_j(w') = \pi_j(w)$.

Ces règles permettent d'appliquer, pour tout $i \in [n]$, les règles de $\mathcal{S}_i$ à la i^{e} projection de tout mot de $\mathscr{L}$ en laissant inchangées les autres projections.

Exemple 4.12 *On considère $n = 2$, $\Omega_1 = \Delta_1 = \{a_1\}$, $\Omega_2 = \Delta_2 = \{a_2\}$, $\mathcal{S}_1 = \{a_1a_1 \to a_1\}$ et $\mathcal{S}_2 = \{a_2 \to a_2a_2\}$. L'ensemble $\mathcal{S}_a$ des règles de réarrangement est*

$$\mathcal{S}_a = \left\{ \binom{\varepsilon}{a_2}\binom{a_1}{\varepsilon} \to \binom{a_1}{a_2}\binom{\varepsilon}{\varepsilon},\ \binom{\varepsilon}{a_2}\binom{a_1}{a_2} \to \binom{a_1}{a_2}\binom{\varepsilon}{a_2},\right.$$

$$\left. \binom{a_1}{\varepsilon}\binom{\varepsilon}{a_2} \to \binom{a_1}{a_2}\binom{\varepsilon}{\varepsilon},\ \binom{a_1}{\varepsilon}\binom{a_1}{a_2} \to \binom{a_1}{a_2}\binom{a_1}{\varepsilon}, \binom{\varepsilon}{\varepsilon} \to \varepsilon \right\}.$$

L'ensemble $\mathcal{S}_s$ des règles de simulation est

$$\mathcal{S}_s = \left\{ \binom{a_1}{\varepsilon}\binom{a_1}{\varepsilon} \to \binom{a_1}{\varepsilon},\ \binom{a_1}{a_2}\binom{a_1}{\varepsilon} \to \binom{a_1}{a_2},\ \binom{a_1}{a_2}\binom{a_1}{a_2} \to \binom{a_1}{a_2}\binom{\varepsilon}{a_2} \right\} \cup$$

$$\left\{ \binom{\varepsilon}{a_2} \to \binom{\varepsilon}{a_2}\binom{\varepsilon}{a_2}, \binom{a_1}{a_2} \to \binom{a_1}{a_2}\binom{\varepsilon}{a_2} \right\}.$$

Le premier (resp. second) membre de cette union correspond aux règles engendrées par $\mathcal{S}_1$ (resp. $\mathcal{S}_2$).

Le langage rationnel R. Pour tout $i \in [n]$, soit $\mathcal{A}_i = \big(Q_i, \Omega_i, \delta_i, q_{0i}, F_i\big)$ un automate fini sans ε-transition qui reconnaît R_i (Q_i est l'ensemble fini des états, Ω_i est l'alphabet de l'automate, $\delta_i \subseteq Q_i \times \Omega_i \times Q_i$ est l'ensemble fini des transitions, $q_{0i} \in Q_i$ est l'état initial et $F_i \subseteq Q_i$ est l'ensemble des états acceptants). L'objectif ici est de construire à partir des $\mathcal{A}_i$ un automate fini qui reconnaît $\phi\big(\prod_{i\in[n]} R_i\big)$, ce qui prouverait que ce langage est régulier. Comme tous les mots de $\phi\big(\prod_{i\in[n]} R_i\big)$ sont irréductibles par les règles de réarrangement (car ϕ est à valeur dans $\mathscr{L}$) et les règles de simulation (comme conséquence de la construction de $\mathcal{S}_s$ et du fait que pour tout $i \in [n]$, $R_i \subseteq \mathrm{Irr}(\mathcal{S}_i)$), on aurait de surcroît l'inclusion de ce langage dans l'ensemble des mots irréductibles par $\mathcal{S}$. On pourrait alors poser $R = \phi\big(\prod_{i\in[n]} R_i\big)$.

Voici comment construire un automate fini qui reconnaît $\phi\big(\prod_{i\in[n]} R_i\big)$. L'idée est d'effectuer un produit des automates $\mathcal{A}_i$. Néanmoins, il est nécessaire dans un premier temps d'ajouter des transitions étiquetées par ε aux automates $\mathcal{A}_i$ si l'on veut que ce produit reconnaisse $\phi\big(\prod_{i\in[n]} R_i\big)$. En effet, si par exemple $n = 2$, $R_1 = \{a\}$ et $R_2 = \{aaa\}$, le mot $(a,a)(\varepsilon,a)(\varepsilon,a)$ est dans $\phi\big(R_1 \times R_2\big)$ car il est égal à $\phi\big(a, aaa\big)$. On associe donc à chaque $\mathcal{A}_i$ l'automate $\mathcal{B}_i$ défini de la façon suivante :

$$\mathcal{B}_i = \big(Q_i \cup \{f'_i\},\ \widetilde{\Omega_i},\ \delta_i \cup \delta'_i,\ q_{0i},\ F_i \cup \{f'_i\}\big) \text{ où :}$$

- f'_i est un nouvel état n'appartenant pas à Q_i et
- $\delta'_i = \big\{\big(f, \varepsilon, f'_i\big) \mid f \in F_i\big\} \cup \big\{(f'_i, \varepsilon, f'_i)\big\}$.

Soit $\mathcal{B}$ le produit des $\mathcal{B}_i$ selon la contrainte $\prod_{i\in[n]} \widetilde{\Omega_i} \smallsetminus \{\vec{\varepsilon}_n\}$[3], c'est-à-dire

$$\mathcal{B} = \Big(\prod_{i\in[n]} Q_i \cup \{f'_i\},\ \prod_{i\in[n]} \widetilde{\Omega_i} \smallsetminus \{\vec{\varepsilon}_n\},\ \delta,\ (q_{01}, \ldots, q_{0n}),\ \prod_{i\in[n]} F_i \cup \{f'_i\}\Big)$$

où δ contient les triplets (q, a, q') tels que $a \neq \vec{\varepsilon}_n$ et, pour tout $i \in [n]$, $\big(\pi_i\big(q\big), \pi_i\big(a\big), \pi_i\big(q'\big)\big) \in \delta_i \cup \delta'_i$. La construction des automates $\mathcal{B}_i$ détaillée ci-dessus assure que le langage reconnu par $\mathcal{B}$ est $\phi\big(\prod_{i\in[n]} R_i\big)$.

Finalement, comme $\phi\big(\prod_{i\in[n]} R_i\big)$ est régulier et, comme expliqué plus haut, est irréductible par $\mathcal{S}$, on pose

$$R = \phi\big(\prod_{i\in[n]} R_i\big)\ .$$

Exemple 4.13 *On considère les langages rationnels $\{\varepsilon, x\}$ et a^*. Voici deux automates finis sans ε-transition qui reconnaissent ces langages :*

$$\mathcal{A}_1 = \Big(\big\{q_0, q_1\big\}, \big\{x\big\}, \big\{(q_0, x, q_1)\big\}, q_0, \big\{q_0, q_1\big\}\Big) \quad \textit{et} \quad \mathcal{A}_2 = \Big(\big\{p_0\big\}, \big\{a\big\}, \big\{(p_0, a, p_0)\big\}, p_0, \big\{p_0\big\}\Big)\ .$$

À partir de $\mathcal{A}_1$ et $\mathcal{A}_2$, on construit :

$$\mathcal{B}_1 = \Big(\big\{q_0, q_1, f'_1\big\}, \big\{x\big\}, \big\{(q_0, x, q_1), (q_0, \varepsilon, f'_1), (q_1, \varepsilon, f'_1), (f'_1, \varepsilon, f'_1)\big\}, q_0, \big\{q_0, q_1, f'_1\big\}\Big) \quad \textit{et}$$

$$\mathcal{B}_2 = \Big(\big\{p_0, f'_2\big\}, \big\{a\big\}, \big\{(p_0, a, p_0), (p_0, \varepsilon, f'_2), (f'_2, \varepsilon, f'_2)\big\}, p_0, \big\{p_0, f'_2\big\}\Big)\ .$$

3. $\vec{\varepsilon}_n$ désigne le n-uplet dont toutes les composantes sont égales au mot vide.

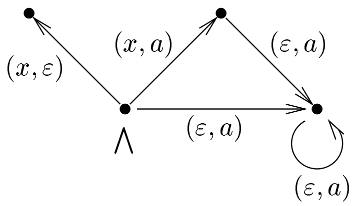
Le produit de $\mathcal{B}_1$ et $\mathcal{B}_2$ selon la contrainte $\big(\{x,\varepsilon\}\times\{a,\varepsilon\}\big)\setminus\{(\varepsilon,\varepsilon)\}$ *a pour*

- *ensemble d'états :* $\big\{(q_0,p_0),(q_1,p_0),(f_1',p_0),(q_0,f_2'),(q_1,f_2'),(f_1',f_2')\big\}$,
- *alphabet :* $\big\{(\varepsilon,a),(x,\varepsilon),(x,a)\big\}$,
- *ensemble de transitions :*

$$\begin{array}{ll}\{ & \big((q_0,p_0),(\varepsilon,a),(f_1',p_0)\big),\big((q_0,p_0),(x,a),(q_1,p_0)\big),\big((q_0,p_0),(x,\varepsilon),(q_1,f_2')\big),\\ & \big((q_1,p_0),(\varepsilon,a),(f_1',p_0)\big),\big((f_1',p_0),(\varepsilon,a),(f_1',p_0)\big),\big((q_0,f_2'),(x,\varepsilon),(q_1,f_2')\big)\quad\},\end{array}$$

- *état initial :* (q_0,p_0),
- *ensemble d'états acceptants :* $\big\{(q_0,p_0),(q_1,p_0),(f_1',p_0),(q_0,f_2'),(q_1,f_2'),(f_1',f_2')\big\}$.

Le langage reconnu par cet automate est $(x,\varepsilon)+(\varepsilon,a)^*+(x,a).(\varepsilon,a)^*$ *c'est-à-dire* $\phi\big(\{x,\varepsilon\}\times a^*\big)$, *ce qui se voit peut-être plus clairement sur le graphe de racine distinguée* (q_0,p_0) *qui lui est associé de façon naturelle (tous les états sont acceptants et le symbole* $\bigwedge$ *désigne l'état initial) :*



Le mot u. Il est défini ainsi : $u=\phi(u_1,\ldots,u_n)$. Il est clair que, comme $R=\phi\big(\prod_{i\in[n]}R_i\big)$, on a $u\in R$.

3.2 Les graphes $\mathcal{G}(\prod^{\mathscr{C}}_{i\in[n]}\mathcal{T}_i)$ et $\prod^{\mathscr{C}}_{i\in[n]}\mathcal{G}(\mathcal{T}_i)$ sont isomorphes

Le principal résultat de cette partie (Théorème 4.17) est démontré ici. Nous établissons auparavant quelques faits qui seront utilisés au cours de la preuve.

Lemme 4.14 *Soit* $\vec{x},\vec{z}\in\prod_{i\in[n]}(\Omega_i\cup\Delta_i)^*$ *tels qu'il existe* $i\in[n]$ *vérifiant* $\pi_i(\vec{x})\xrightarrow[\mathcal{S}_i]{*}\pi_i(\vec{z})$ *et pour tout* $j\neq i$, $\pi_j(\vec{z})=\pi_j(\vec{x})$. *Alors,* $\phi(\vec{x})\xrightarrow[\mathcal{S}]{*}\phi(\vec{z})$.

Preuve. Nous procédons par récurrence sur la longueur de la réécriture $\pi_i(\vec{x})\xrightarrow[\mathcal{S}_i]{*}\pi_i(\vec{z})$.

- *Cas de base.* Si cette longueur est nulle, le résultat est immédiat.
- *Récurrence.* Supposons que lorsque la longueur de la réécriture est $m\geq 0$ fixé quelconque, le résultat du lemme est vérifié. Maintenant, si cette longueur est $m+1$, il existe $y_i\in(\Omega_i\cup\Delta_i)^*$ tel que $\pi_i(\vec{x})\xrightarrow[\mathcal{S}_i]{}y_i\xrightarrow[\mathcal{S}_i]{m}\pi_i(\vec{z})$. Soit $\vec{y}$ le n-uplet défini par : $\pi_i(\vec{y})=y_i$ et $\pi_j(\vec{y})=\pi_j(\vec{x})$ pour tout $j\neq i$. Alors, on a $\pi_i(\vec{x})\xrightarrow[\mathcal{S}_i]{}\pi_i(\vec{y})\xrightarrow[\mathcal{S}_i]{m}\pi_i(\vec{z})$.

Le mot $\pi_i(\vec{x})$ est réductible par une règle $l_i \to r_i \in \mathcal{S}_i$, donc $\phi(\vec{x})$ contient un facteur[4] w tel que $\pi_i(w) = l_i$. Bien entendu, comme tout facteur d'un mot de $\mathscr{L}$ est un mot de $\mathscr{L}$, on a $w \in \mathscr{L}$ et on peut supposer que $\forall j \in [\ |w|\ ]$, $\pi_i(w(j)) \neq \varepsilon$. De plus, par construction, $\mathcal{S}_s$ contient une règle $w \to w'$ telle que $\pi_i(w') = r_i$ et $\forall j \in [n] \setminus \{i\}$, $\pi_j(w') = \pi_j(w)$.

Appelons y le mot qui est obtenu en appliquant la règle $w \to w'$ au mot $\phi(\vec{x})$. Alors, y est tel que $\pi_i(y) \overset{\text{déf.}\mathcal{S}_s}{=} y_i \overset{\text{déf.}\vec{y}}{=} \pi_i(\vec{y})$ et $\forall j \in [n] \setminus \{i\}$, $\pi_j(y) \overset{\text{déf.}\mathcal{S}_s}{=} \pi_j(\phi(\vec{x})) \overset{\text{déf.}\phi}{=} \pi_j(\vec{x}) \overset{\text{déf.}\vec{y}}{=} \pi_j(\vec{y})$. Donc, $\forall j \in [n]$, $\pi_j(y) = \pi_j(\vec{y})$.

Soit $y' = y{\downarrow_{\mathcal{S}_a}}$. D'après le lemme 4.11, on a $\forall j \in [n]$, $\pi_j(y') = \pi_j(y)$ et $y' \in \mathscr{L}$, et donc, par définition de ϕ, $y' = \phi(\vec{y})$. Par conséquent, $\phi(\vec{x}) \xrightarrow[\mathcal{S}]{*} \phi(\vec{y})$. Finalement, par hypothèse de récurrence, $\phi(\vec{x}) \xrightarrow[\mathcal{S}]{*} \phi(\vec{z})$. □

Proposition 4.15 *Soit $\vec{x}, \vec{y} \in \prod_{i\in[n]} R_i$ et $\vec{a} \in \mathscr{C}$ tels que pour tout $i \in [n]$, on a la réécriture $\pi_i(\vec{x}).\pi_i(\vec{a}) \xrightarrow[\mathcal{S}_i]{*} \pi_i(\vec{y})$. Alors, $\phi(\vec{x}).\vec{a} \xrightarrow[\mathcal{S}]{*} \phi(\vec{y})$.*

Preuve. On note $\oplus$ l'opération de concaténation habituelle pour les produits de monoïdes, *i.e.* pour tous vecteurs $(w_1, \dots, w_n)$ et $(w'_1, \dots, w'_n)$ dans $\prod_{i\in[n]}(\Omega_i \cup \Delta_i)^*$, on a

$$(w_1, \dots, w_n) \oplus (w'_1, \dots, w'_n) = (w_1.w'_1, \dots, w_n.w'_n)\ .$$

Soit $(\vec{x}_i)_{i\leq n}$ la suite définie ainsi :

- $\vec{x}_0 = \vec{x} \oplus \vec{a}$ et
- pour tout $i \in [n]$, $\vec{x}_i$ est le n-uplet défini par $\pi_i(\vec{x}_i) = \pi_i(\vec{y})$ et pour tout $j \neq i$, $\pi_j(\vec{x}_i) = \pi_j(\vec{x}_{i-1})$.

Pour tout $i \in [n]$, $\pi_i(\vec{x}_{i-1}) = \pi_i(\vec{x}_0) = \pi_i(\vec{x} \oplus \vec{a}) = \pi_i(\vec{x}).\pi_i(\vec{a})$. Par conséquent, on a la réécriture $\pi_i(\vec{x}_{i-1}) \xrightarrow[\mathcal{S}_i]{*} \pi_i(\vec{y})$ *i.e.* $\pi_i(\vec{x}_{i-1}) \xrightarrow[\mathcal{S}_i]{*} \pi_i(\vec{x}_i)$. De plus, pour tout $j \neq i$, on a $\pi_j(\vec{x}_{i-1}) = \pi_j(\vec{x}_i)$. Donc, d'après le lemme 4.14, pour tout $i \in [n]$, on a $\phi(\vec{x}_{i-1}) \xrightarrow[\mathcal{S}]{*} \phi(\vec{x}_i)$. Comme $\vec{x}_0 = \vec{x} \oplus \vec{a}$ et $\vec{x}_n = \vec{y}$, on a finalement $\phi(\vec{x} \oplus \vec{a}) \xrightarrow[\mathcal{S}]{*} \phi(\vec{y})$. Puisque $\phi(\vec{x}).\phi(\vec{a}) \xrightarrow[\mathcal{S}_a]{*} \phi(\vec{x} \oplus \vec{a})$ et $\phi(\vec{a}) = \vec{a}$, on obtient $\phi(\vec{x}).\vec{a} \xrightarrow[\mathcal{S}]{*} \phi(\vec{y})$. □

Proposition 4.16 *Soit $\vec{x}, \vec{y} \in \prod_{i\in[n]} R_i$ et $\vec{a} \in \mathscr{C}$ tels que $\phi(\vec{x}).\vec{a} \xrightarrow[\mathcal{S}]{*} \phi(\vec{y})$. Alors, pour chaque $i \in [n]$, $\pi_i(\vec{x}).\pi_i(\vec{a}) \xrightarrow[\mathcal{S}_i]{*} \pi_i(\vec{y})$.*

Preuve. Soit $w_0 \xrightarrow[\mathcal{S}]{} w_1 \xrightarrow[\mathcal{S}]{} \dots \xrightarrow[\mathcal{S}]{} w_p$ une réécriture de $\phi(\vec{x}).\vec{a}$ en $\phi(\vec{y})$ (*i.e.* $w_0 = \phi(\vec{x}).\vec{a}$ et $w_p = \phi(\vec{y})$). L'application d'une règle de $\mathcal{S}_s$ à un mot w_k de la réécriture correspond à l'application d'une règle d'un $\mathcal{S}_i$ à $\pi_i(w_k)$ et ne modifie pas les mots $\pi_j(w_k)$, $j \in [n] \setminus \{i\}$. De plus, les règles de réarrangement ne modifient pas les mots $\pi_j(w_k)$, $j \in [n]$. Par conséquent, w_p est tel que pour tout $i \in [n]$, $\pi_i(w_0) \xrightarrow[\mathcal{S}_i]{*} \pi_i(w_p)$. Or, $\pi_i(w_0) = \pi_i(\phi(\vec{x}).\vec{a}) = \pi_i(\phi(\vec{x})).\pi_i(\vec{a})$, $\pi_i(\phi(\vec{x})) \overset{\text{déf.}\phi}{=} \pi_i(\vec{x})$ et $\pi_i(\phi(\vec{y})) \overset{\text{déf.}\phi}{=} \pi_i(\vec{y})$. Par conséquent, le résultat de la proposition est vérifié. □

Nous sommes à présent en mesure de prouver le théorème suivant.

Théorème 4.17 *Les graphes $\mathcal{G}(\prod^{\mathscr{C}}_{i\in[n]} \mathcal{T}_i)$ et $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{T}_i)$ sont isomorphes.*

4. Nous rappelons au lecteur qu'un mot w' est un *facteur* d'un mot $w \neq \varepsilon$ si $w' = \varepsilon$ ou s'il existe deux entiers naturels i et j tels que $1 < i < j < |w|$ et $w' = w(i) \dots w(j)$. Si w est le mot vide, son seul facteur est lui-même.

Preuve. On a : ϕ est une bijection, $R = \phi(\prod_{i\in[n]} R_i)$, $u = \phi(u_1, \ldots, u_n)$ et les graphes $\mathcal{G}(\prod^{\mathscr{C}}_{i\in[n]} \mathcal{T}_i)$ et $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{T}_i)$ sont étiquetés par $\mathscr{C}$. De plus, étant données les propositions 4.15 et 4.16, pour tout $a \in \mathscr{C}$, $d \xrightarrow{a} d'$ est un arc de $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{T}_i)$ si et seulement si $\phi(d) \xrightarrow{a} \phi(d')$ est un arc de $\mathcal{G}(\prod^{\mathscr{C}}_{i\in[n]} \mathcal{T}_i)$. $\square$

3.3 Complexité de la construction

Taille de l'automate $\mathcal{B}$. Le nombre d'états et de transitions de l'automate $\mathcal{B}$ sont égaux respectivement à $\prod_{i\in[n]} \big(|Q_i| + 1\big)$ et $\prod_{i\in[n]} \big(|\delta_i| + |F_i| + 1\big) - \prod_{i\in[n]} \big(|F_i| + 1\big)$. Les nombres $|F_i| + 1$ correspondent aux ε-transitions que l'on rajoute aux automates $\mathcal{A}_i$. De plus, il ne faut pas considérer, lorsqu'on construit $\mathcal{B}$, les transitions étiquetées par le vecteur $\vec{\varepsilon}_n$.

Taille du semi-système de Thue $\mathcal{S}$. Dans le discours qui suit, pour tout $i \in [n]$, on note n_i le nombre $|\Omega_i \cup \Delta_i|$.

- *Nombre de règles de réarrangement* qui ont la forme $\vec{t}_1\vec{t}_2 \to \vec{u}_1\vec{u}_2$. Le choix d'un couple $(\vec{t}_1, \vec{t}_2)$ détermine un unique couple $(\vec{u}_1, \vec{u}_2)$ vérifiant les propriétés données dans la définition de $\mathcal{S}_a$. Donc, pour compter le nombre de ces règles, on s'intéresse au nombre de choix possibles de couples $(\vec{t}_1, \vec{t}_2)$.

  Soit $k \in [n-1]$. On note $\mathbf{P}^k_n$ l'ensemble des parties de $[n]$ qui contiennent k éléments. Soit P un élément de $\mathbf{P}^k_n$. Nous rappelons que toute règle de réarrangement qui a la forme $\vec{t}_1\vec{t}_2 \to \vec{u}_1\vec{u}_2$ vérifie : $\vec{t}_1 \neq \vec{\varepsilon}_n$ et si $S = \{i \in [n] \mid \pi_i\big(\vec{t}_1\big) = \varepsilon$ et $\pi_i\big(\vec{t}_2\big) \neq \varepsilon\}$, alors $S \neq \varnothing$ et

  - $\forall i \in S$, $\pi_i\big(\vec{u}_2\big) = \varepsilon$ et $\pi_i\big(\vec{u}_1\big) = \pi_i\big(\vec{t}_2\big)$ et
  - $\forall i \notin S$, $\pi_i\big(\vec{u}_1\big) = \pi_i\big(\vec{t}_1\big)$ et $\pi_i\big(\vec{u}_2\big) = \pi_i\big(\vec{t}_2\big)$.

  Nous allons calculer le nombre de couples $(\vec{t}_1, \vec{t}_2)$ qu'il est possible de former en ayant $S = P$, c'est-à-dire en ayant

  1. $\forall p \in P,\ \pi_p\big(\vec{t}_1\big) = \varepsilon \wedge \pi_p\big(\vec{t}_2\big) \neq \varepsilon$ et
  2. $\forall \overline{p} \notin P,\ \Big(\pi_{\overline{p}}\big(\vec{t}_1\big) = \varepsilon \wedge \pi_{\overline{p}}\big(\vec{t}_2\big) = \varepsilon\Big) \vee \Big(\pi_{\overline{p}}\big(\vec{t}_1\big) \in \Omega_i \cup \Delta_i \wedge \pi_{\overline{p}}\big(\vec{t}_2\big) \in \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\Big)$.

  Pour un P donné, il y a, d'après l'énoncé 1 ci-dessus, $\prod_{p\in P} n_p$ façons de choisir les coordonnées de $\vec{t}_1$ et $\vec{t}_2$ appartenant à P et il y a, d'après l'énoncé 2,

  $$\prod_{\overline{p}\in[n]\smallsetminus P} \big(1 + n_{\overline{p}}.(n_{\overline{p}} + 1)\big) - 1$$

  façons de choisir les coordonnées de $\vec{t}_1$ et $\vec{t}_2$ n'appartenant pas à P (le -1 correspond au cas interdit où $\vec{t}_1 \in \prod_{i\in[n]}\{\varepsilon\}$). Pour un P donné, il y a donc

  $$\prod_{p\in P} n_p.\Big[\prod_{\overline{p}\in[n]\smallsetminus P} \big(1 + n_{\overline{p}}.(n_{\overline{p}} + 1)\big) - 1\Big]$$

  façons de former un couple $(\vec{t}_1, \vec{t}_2)$ en ayant $S = P$.

Le nombre total de règles de réarrangement qui ont la forme $\vec{t}_1\vec{t}_2 \to \vec{u}_1\vec{u}_2$ s'obtient alors en considérant pour tout $k \in [n-1]$ l'ensemble des parties de $[n]$ à k éléments. Ce nombre est

$$\sum_{k\in[n-1]} \sum_{P\in\mathbf{P}_n^k} \left(\prod_{p\in P} n_p . \Big[\prod_{\overline{p}\in[n]\setminus P} \big(1 + n_{\overline{p}}.(n_{\overline{p}}+1)\big) - 1 \Big] \right) .$$

Dans le cas où les n_i sont tous égaux, le nombre de ces règles est

$$\sum_{p\in[n-1]} \mathbf{C}_n^p . n_1^p . \big[(1 + n_1.(n_1+1))^{n-p} - 1\big]$$

(où $\mathbf{C}_n^p = \frac{n!}{p!(n-p)!}$ est le nombre de combinaisons de p éléments de $[n]$).

– *Nombre de règles de simulation.* Soit $i \in [n]$ et $l_i \to r_i$ une règle de $\mathcal{S}_i$. Pour chaque $k \in [n] \setminus \{i\}$, la seule façon de « transformer » cette règle en une règle de simulation est de considérer tous les $|l_i|$-uplets $\big(\pi_k(\vec{t}_1), \ldots, \pi_k(\vec{t}_{|l_i|})\big)$ tels qu'il existe $p \in [\,|l_i|\,]$ vérifiant $\{\pi_k(\vec{t}_1), \ldots, \pi_k(\vec{t}_p)\} \subseteq \Omega_k \cup \Delta_k$ et $\{\pi_k(\vec{t}_{p+1}), \ldots, \pi_k(\vec{t}_{|l_i|})\} \subseteq \{\varepsilon\}$ (ceci bien sûr dans le but d'obtenir une règle de simulation dont les membres gauche et droit sont dans $\mathscr{L}$). Pour un i donné, le nombre de ces $|l_i|$-uplets est

$$1 + \sum_{p\in[\,|l_i|\,]} {n_k}^p .$$

Par conséquent, pour chaque $i \in [n]$ et chaque $l_i \to r_i \in \mathcal{S}_i$, il y a

$$\prod_{k\in[n]\setminus\{i\}} \Big(1 + \sum_{p\in[\,|l_i|\,]} {n_k}^p\Big)$$

règles possibles. Donc, au total, le nombre de règles de simulation est :

$$\sum_{i\in[n]} \left[\sum_{l_i \to r_i\in\mathcal{S}_i} \left(\prod_{k\in[n]\setminus\{i\}} \Big(1 + \sum_{p\in[\,|l_i|\,]} {n_k}^p\Big) \right) \right] .$$

Dans le cas où les n_i sont tous égaux, le nombre de ces règles est

$$\sum_{i\in[n]} \sum_{l_i \to r_i\in\mathcal{S}_i} \Big(1 + \sum_{p\in[\,|l_i|\,]} {n_1}^p\Big)^{n-1} .$$

Toutes ces formules permettent d'énoncer la proposition suivante.

Proposition 4.18 *Le nombre de règles du semi-système de Thue de notre construction est exponentiel en le nombre de spécifications qu'on s'est données au départ, exponentiel en la taille des membres gauches de règle des semi-systèmes de Thue de ces spécifications et polynomial en la taille de leurs alphabets.*

3.4 Un exemple complet

On considère les deux spécifications

$$\mathcal{T}_1 = \Big(\{x,y\},\{x,y\},\{xy \to \varepsilon\},\{\varepsilon,x\},\varepsilon\Big) \text{ et}$$

$$\mathcal{T}_2 = \Big(\{a,b\},\{a,b\},\{b \to a\},a^*,\varepsilon\Big)\ .$$

Leurs graphes sont

$$\begin{tikzcd} \boxed{\varepsilon} \arrow[r, bend left, "x"] & \boxed{x} \arrow[l, bend left, "y"] \end{tikzcd} \qquad \text{et} \qquad \boxed{\varepsilon} \xrightarrow{a,b} \boxed{a} \xrightarrow{a,b} \boxed{aa} \ \text{- - - -}$$

Soit $\mathscr{C} = \{(x,a),(y,b)\}$ une contrainte de synchronisation. Le produit des deux graphes ci-dessus selon $\mathscr{C}$ est :

$$\boxed{(\varepsilon,\varepsilon)} \xrightarrow{(x,a)} \boxed{(x,a)} \xrightarrow{(y,b)} \boxed{(\varepsilon,aa)} \ \text{- - - - - -}$$

Nous construisons à partir de $\mathcal{T}_1$ et $\mathcal{T}_2$ la spécification $\mathcal{T}$ comme indiqué précédemment. D'après les formules données au paragraphe précédent, $\mathcal{T}$ est composée de

$$\sum_{P\in\{\{1\},\{2\}\}} \Big(\prod_{i\in P} n_i.\Big[\prod_{i\in[2]\smallsetminus P}\big(1+n_i.(n_i+1)\big)-1\Big]\Big) =$$

$$n_1.\big[(1+n_2.(n_2+1))-1\big]+n_2.\big[(1+n_1.(n_1+1))-1\big] =$$

$$2.\big[(1+2.(2+1))-1\big]+2.\big[(1+2.(2+1))-1\big] = 24$$

règles de réarrangement ayant la forme $\vec{t}_1\vec{t}_2 \to \vec{u}_1\vec{u}_2$. Le nombre de règles de simulation est quant à lui égal à

$$\sum_{i\in[2]}\Big[\sum_{l_i\to r_i\in\mathcal{S}_i}\Big(\prod_{k\in[2]\smallsetminus\{i\}}\Big(1+\sum_{p\in[\,|l_i|\,]} {n_k}^p\Big)\Big)\Big] = (1+n_2+n_2^2)+(1+n_1) = 10\ .$$

Les règles de réarrangement sont :

$$\begin{aligned}
\mathcal{S}_a = \Big\{ \; & \binom{\varepsilon}{a}\binom{x}{\varepsilon} \to \binom{x}{a}\binom{\varepsilon}{\varepsilon}, \binom{\varepsilon}{a}\binom{x}{a} \to \binom{x}{a}\binom{\varepsilon}{a}, \binom{\varepsilon}{a}\binom{x}{b} \to \binom{x}{a}\binom{\varepsilon}{b}, \\
& \binom{\varepsilon}{a}\binom{y}{\varepsilon} \to \binom{y}{a}\binom{\varepsilon}{\varepsilon}, \binom{\varepsilon}{a}\binom{y}{a} \to \binom{y}{a}\binom{\varepsilon}{a}, \binom{\varepsilon}{a}\binom{y}{b} \to \binom{y}{a}\binom{\varepsilon}{b}, \\
& \binom{\varepsilon}{b}\binom{x}{\varepsilon} \to \binom{x}{b}\binom{\varepsilon}{\varepsilon}, \binom{\varepsilon}{b}\binom{x}{a} \to \binom{x}{b}\binom{\varepsilon}{a}, \binom{\varepsilon}{b}\binom{x}{b} \to \binom{x}{b}\binom{\varepsilon}{b}, \\
& \binom{\varepsilon}{b}\binom{y}{\varepsilon} \to \binom{y}{b}\binom{\varepsilon}{\varepsilon}, \binom{\varepsilon}{b}\binom{y}{a} \to \binom{y}{b}\binom{\varepsilon}{a}, \binom{\varepsilon}{b}\binom{y}{b} \to \binom{y}{b}\binom{\varepsilon}{b}, \\
& \binom{x}{\varepsilon}\binom{\varepsilon}{a} \to \binom{x}{a}\binom{\varepsilon}{\varepsilon}, \binom{x}{\varepsilon}\binom{x}{a} \to \binom{x}{a}\binom{x}{\varepsilon}, \binom{x}{\varepsilon}\binom{y}{a} \to \binom{x}{a}\binom{y}{\varepsilon}, \\
& \binom{x}{\varepsilon}\binom{\varepsilon}{b} \to \binom{x}{b}\binom{\varepsilon}{\varepsilon}, \binom{x}{\varepsilon}\binom{x}{b} \to \binom{x}{b}\binom{x}{\varepsilon}, \binom{x}{\varepsilon}\binom{y}{b} \to \binom{x}{b}\binom{y}{\varepsilon}, \\
& \binom{y}{\varepsilon}\binom{\varepsilon}{a} \to \binom{y}{a}\binom{\varepsilon}{\varepsilon}, \binom{y}{\varepsilon}\binom{x}{a} \to \binom{y}{a}\binom{x}{\varepsilon}, \binom{y}{\varepsilon}\binom{y}{a} \to \binom{y}{a}\binom{y}{\varepsilon}, \\
& \binom{y}{\varepsilon}\binom{\varepsilon}{b} \to \binom{y}{b}\binom{\varepsilon}{\varepsilon}, \binom{y}{\varepsilon}\binom{x}{b} \to \binom{y}{b}\binom{x}{\varepsilon}, \binom{y}{\varepsilon}\binom{y}{b} \to \binom{y}{b}\binom{y}{\varepsilon}, \\
& \binom{\varepsilon}{\varepsilon} \to \varepsilon \;\Big\}
\end{aligned}$$

et les règles de simulation sont :

$$\begin{aligned}
\mathcal{S}_s = \Big\{ \; & \binom{x}{\varepsilon}\binom{y}{\varepsilon} \to \binom{\varepsilon}{\varepsilon}, \\
& \binom{x}{a}\binom{y}{\varepsilon} \to \binom{\varepsilon}{a}, \binom{x}{a}\binom{y}{a} \to \binom{\varepsilon}{a}\binom{\varepsilon}{a}, \binom{x}{a}\binom{y}{b} \to \binom{\varepsilon}{a}\binom{\varepsilon}{b}, \\
& \binom{x}{b}\binom{y}{\varepsilon} \to \binom{\varepsilon}{b}, \binom{x}{b}\binom{y}{a} \to \binom{\varepsilon}{b}\binom{\varepsilon}{a}, \binom{x}{b}\binom{y}{b} \to \binom{\varepsilon}{b}\binom{\varepsilon}{b}, \\
& \binom{\varepsilon}{b} \to \binom{\varepsilon}{a}, \binom{x}{b} \to \binom{x}{a}, \binom{y}{b} \to \binom{y}{a} \;\Big\} .
\end{aligned}$$

Le langage R est celui reconnu par le produit des automates $\mathcal{B}_1$ et $\mathcal{B}_2$ de l'exemple 4.13, *i.e.* :

$$R = \binom{x}{\varepsilon} + \binom{\varepsilon}{a}^* + \binom{x}{a}\binom{\varepsilon}{a}^* .$$

Enfin, le mot u est $\phi(\varepsilon, \varepsilon) = \varepsilon$.

Le graphe de la spécification dont chaque composante vient d'être définie est :

$$\boxed{\varepsilon} \xrightarrow{\quad \binom{x}{a} \quad} \boxed{\binom{x}{a}} \xrightarrow{\quad \binom{y}{b} \quad} \boxed{\binom{\varepsilon}{a}\binom{\varepsilon}{a}} \quad ------$$

Il est isomorphe au produit synchronisé de $\mathcal{G}(\mathcal{T}_1)$ et $\mathcal{G}(\mathcal{T}_2)$ selon $\mathscr{C}$.

Chapitre 5

Graphes des machines de Turing

Sommaire

La classe des graphes finis est liée aux langages rationnels et celle des graphes de machines à pile aux langages algébriques. Ces deux classes de graphes ont déjà été étudiées mais on ne sait pratiquement rien au sujet des graphes liés aux niveaux suivants de la hiérarchie de Chomsky, c'est-à-dire les langages sensibles au contexte et les langages récursivement énumérables.

Dans ce chapitre, nous étudions les graphes des machines de Turing en nous concentrant sur des aspects liés à la spécification et à la vérification de processus. Certaines des propriétés que nous exposons sont une généralisation des résultats que nous avons présentés dans [KP99] et qui concernent les machines linéairement bornées. Nous étudions la propriété de fermeture de la classe des graphes des machines de Turing par produit synchronisé et nous examinons la question de la décidabilité de la théorie du premier ordre de ces graphes.

La partie la plus volumineuse de ce chapitre concerne les liens qui unissent les graphes des machines de Turing à ceux des spécifications de Thue. Nous fournissons une construction qui permet d'associer une machine de Turing à une spécification de Thue et une deuxième qui permet d'effectuer l'association inverse.

1 Définitions

1.1 Machines de Turing

Nous utilisons dans ce chapitre une définition de machines de Turing largement inspirée de celle des machines de Turing « off-line » (voir par exemple [MS97] pour plus de détails). Une machine de Turing est composée d'une *bande d'entrée* qu'elle peut lire mais sur laquelle elle ne peut pas écrire, ainsi que de *bandes de travail*. Tous les calculs sont effectués sur ces dernières qui sont infinies à droite et à gauche. À l'instant initial, un *mot d'entrée* est inscrit sur la bande d'entrée et les bandes de travail ne contiennent que des *blancs*. Ensuite, la tête de lecture de la bande d'entrée lit le mot d'entrée de la gauche vers la droite en s'arrêtant parfois s'il y a besoin et des calculs sont effectués sur les bandes de travail. La tête de lecture de la bande d'entrée de ces machines fonctionne donc toujours dans le même sens, de gauche à droite, contrairement à celle des machines de Turing « off-line » traditionnelles. De plus, il n'y a pas d'états acceptants. Ceci est dû au fait que nous ne sommes pas intéressés par les propriétés des machines de Turing liées à la théorie des langages. Nous considérons plutôt celles-ci comme une technique de modélisation du comportement de processus grâce au graphe associé à la machine.

De manière plus formelle, une machine de Turing $\mathcal{M}$ à k bandes de travail et sur l'alphabet Σ est un uplet $(Q, \Sigma, \Gamma_1, \ldots, \Gamma_k, \delta, q_0)$ où Q est l'ensemble fini des *états*, Σ est *l'alphabet d'entrée*, Γ_1, ... , Γ_k sont les *alphabets des bandes de travail*, q_0 est *l'état initial*, et δ est l'ensemble des *transitions* :

$$\delta \subseteq Q \times \widetilde{\Sigma} \times \Gamma_1 \times \cdots \times \Gamma_k \times \Gamma_1 \times \{◀, ▶, ■\} \times \cdots \times \Gamma_k \times \{◀, ▶, ■\} \times Q$$

où ◀ (resp. ▶ et ■) représente un mouvement vers la gauche (resp. vers la droite et pas de mouvement) sur une bande de travail et l'entrée ε représente le cas où la tête de lecture de la bande d'entrée ne bouge pas et ne lit rien. Nous supposons que le caractère blanc, noté $\square$, appartient à tous les Γ_i.

Une *ε-transition* est un élément de l'ensemble

$$Q \times \{\varepsilon\} \times \Gamma_1 \times \cdots \times \Gamma_k \times \Gamma_1 \times \{◀, ▶, ■\} \times \cdots \times \Gamma_k \times \{◀, ▶, ■\} \times Q \ .$$

Une *configuration interne* $(\mu_1 q \nu_1, \ldots, \mu_k q \nu_k)$ de $\mathcal{M}$ est un élément de l'ensemble $\Gamma_1{}^*.Q.\Gamma_1{}^* \times \cdots \times \Gamma_k{}^*.Q.\Gamma_k{}^*$ et donne une description de la machine à un instant donné : q est l'état courant, pour tout i dans $[k]$, $\mu_i \nu_i$ est le contenu de la i^{e} bande de travail depuis caractère différent du blanc le plus à gauche jusqu'au caractère différent du blanc le plus à droite et pour tout i dans $[k]$, la tête de lecture/écriture de la i^{e} bande de travail lit le premier caractère de ν_i ou bien le blanc si ν_i est vide. Notons que pour tout i dans $[k]$, μ_i et ν_i peuvent contenir des caractères blancs. Une configuration interne dont toutes les coordonnées valent q_0 est appelée *configuration initiale* de $\mathcal{M}$.

Une machine de Turing à k bandes de travail est *linéairement bornée* s'il existe k fonctions linéaires S_1, ... , S_k de $\mathbb{N}$ dans $\mathbb{N}$ telles que, lorsque la machine a terminé son travail à partir d'une entrée de longueur n, elle a utilisé au plus $S_i(n)$ cellules sur la i^{e} bande de travail.

Enfin, une machine de Turing est *temps-réel* si elle n'a aucune ε-transition.

1.2 Graphe d'une machine de Turing

Soit $\mathcal{M} = (Q, \Sigma, \Gamma_1, \ldots, \Gamma_k, \delta, q_0)$ une machine de Turing. Soit $\mathcal{G}$ le graphe défini comme suit. Les sommets de $\mathcal{G}$ sont toutes les configurations internes de $\mathcal{M}$, les étiquettes de $\mathcal{G}$ appartiennent à $\widetilde{\Sigma}$ et $(\mu_1 q \nu_1, \ldots, \mu_k q \nu_k) \xrightarrow[\mathcal{G}]{c} (\mu'_1 q' \nu'_1, \ldots, \mu'_k q' \nu'_k)$ si et seulement si pour chaque $i \in [k]$, il existe $X_i, Y_i \in \Gamma_i$ et $\blacklozenge_i \in \{\blacktriangleleft, \blacktriangleright, \blacksquare\}$ tels que

$$(q, c, X_1, \ldots, X_k, Y_1, \blacklozenge_1, \ldots, Y_k, \blacklozenge_k, q') \in \delta \quad \text{et}$$

ou bien $\blacklozenge_i = \blacktriangleleft$ et l'un des points suivants est vérifié :

- $\exists \alpha_i, \beta_i \in {\Gamma_i}^*$, $\exists Z_i \in \Gamma_i$, $\mu_i = \alpha_i Z_i$, $\nu_i = X_i \beta_i$, $\mu'_i = \alpha_i$ et, si $Z_i Y_i \beta_i \neq \Box\Box$, alors $\nu'_i = Z_i Y_i \beta_i$, sinon $\nu'_i = \varepsilon$;
- $\mu_i = \varepsilon$, $\exists \alpha_i \in {\Gamma_i}^*$, $\nu_i = X_i \alpha_i$, $\mu'_i = \varepsilon$ et, si $Y_i \alpha_i \neq \Box$, alors $\nu'_i = \Box Y_i \alpha_i$, sinon $\nu'_i = \varepsilon$;
- $X_i = \Box$, $\exists \alpha_i \in {\Gamma_i}^*$, $\exists Z_i \in \Gamma_i$, $\mu_i = \alpha_i Z_i$, $\nu_i = \varepsilon$, $\mu'_i = \alpha_i$ et, si $Z_i Y_i \neq \Box\Box$, alors $\nu'_i = Z_i Y_i$, sinon $\nu'_i = \varepsilon$;
- $X_i = \Box$, $\mu_i = \varepsilon$, $\nu_i = \varepsilon$, $\mu'_i = \varepsilon$ et, si $Y_i \neq \Box$, alors $\nu'_i = \Box Y_i$, sinon $\nu'_i = \varepsilon$;

ou bien $\blacklozenge_i = \blacktriangleright$ et l'un des points suivants est vérifié :

- $\exists \alpha_i \in {\Gamma_i}^*$, $\nu_i = X_i \alpha_i$, $\nu'_i = \alpha_i$ et, si $\mu_i Y_i \neq \Box$, alors $\mu'_i = \mu_i Y_i$ sinon $\mu'_i = \varepsilon$;
- $X_i = \Box$, $\nu_i = \varepsilon$, $\nu'_i = \varepsilon$ et, si $\mu_i Y_i \neq \Box$, alors $\mu'_i = \mu_i Y_i$, sinon $\mu'_i = \varepsilon$;

ou bien $\blacklozenge_i = \blacksquare$ et l'un des points suivants est vérifié :

- $\exists \alpha_i \in {\Gamma_i}^*$, $\nu_i = X_i \alpha_i$, $\mu'_i = \mu_i$ et, si $Y_i \alpha_i \neq \Box$, alors $\nu'_i = Y_i \alpha_i$, sinon $\nu'_i = \varepsilon$;
- $X_i = \Box$, $\nu_i = \varepsilon$, $\mu'_i = \mu_i$ et, si $Y_i \neq \Box$, alors $\nu'_i = Y_i$, sinon $\nu'_i = \varepsilon$.

Le graphe de $\mathcal{M}$, noté $\mathcal{G}(\mathcal{M})$, est défini comme étant le couple $(\mathcal{G}', \iota)$ [5] où ι est la configuration initiale de $\mathcal{M}$ et $\mathcal{G}'$ est le sous-graphe maximal de $\mathcal{G}$ dont ι est une racine.

On note $\mathcal{G}$[TM] la classe des graphes des machines de Turing et $\mathcal{G}$[Lin-TM] et $\mathcal{G}$[Real-TM] ses sous-classes correspondant aux graphes des machines linéairement bornées et des machines temps-réel.

2 Des machines à plusieurs bandes aux machines à une seule bande

Dans cette partie, nous montrons que toute machine de Turing à k bandes de travail est ε-équivalente à une machine à une seule bande de travail. Ce résultat n'est pas forcément une conséquence du fait que les langages reconnus par les machines de Turing à k bandes sont les mêmes que ceux reconnus par les machines de Turing à une bande de travail. En effet, lorsque

5. Cette notation est définie au chapitre 2 et signifie que l'on distingue dans $\mathcal{G}'$ la racine ι.

deux types de machines reconnaissent exactement les mêmes classes de langages, les classes de graphes qu'elles engendrent ne sont pas toujours ε-équivalentes. Par exemple, les automates à piles (c'est-à-dire les machines à pile pourvues d'états acceptants) et les automates à pile temps-réel reconnaissent exactement les langages algébriques. Pourtant, les graphes des machines à pile ne sont pas ε-équivalents à ceux des machines à pile temps-réel en général. En effet, il est prouvé dans [CK99b] que tout graphe préfixe-reconnaissable ayant une racine distinguée est ε-équivalent à un graphe de machine à pile mais n'est pas, en général, ε-équivalent à un graphe de machine à pile temps-réel.

Soit $\mathcal{M} = (Q, \Sigma, \Gamma_1, \ldots, \Gamma_k, \delta, q_0)$ une machine de Turing à k bandes de travail. On construit la machine $\mathcal{M}'$ à une seule bande de travail de la façon suivante. D'une part, l'inscription de la bande de travail de $\mathcal{M}'$ est la concaténation des inscriptions de toutes les bandes de travail de $\mathcal{M}$ séparées par des *délimiteurs*. D'autre part, le mouvement des têtes de lecture/écriture des bandes de travail de $\mathcal{M}$ est simulé par $\mathcal{M}'$ au moyen de *marqueurs de tête*.

Nous introduisons donc les nouveaux caractères suivants : pour tout i dans $[k+1]$, un délimiteur $\%_i$, et, pour chaque lettre X de $\cup_{i\in[k]}\Gamma_i$, un marqueur de tête $\dot{X}$. Dans la suite, pour tout i dans $[k]$, $\dot{\Gamma}_i$ représente l'ensemble $\{\dot{X} \mid X \in \Gamma_i\}$. Une configuration interne $(\mu_1 q \nu_1, \ldots, \mu_k q \nu_k)$ de $\mathcal{M}$ est simulée par une inscription $\%_1\ \alpha_1 \dot{X}_1 \beta_1\ \%_2\ \ldots\ \%_k\ \alpha_k \dot{X}_k \beta_k\ \%_{k+1}$ sur la bande de travail de $\mathcal{M}'$. Cette inscription est telle que, pour chaque i dans $[k]$, $\nu_i \neq \varepsilon \Rightarrow (\exists \gamma_i \in \{\Box\}^*,\ \nu_i\gamma_i = X_i\beta_i)$, $\nu_i = \varepsilon \Rightarrow (X_i = \Box$ et $\beta_i \in \{\Box\}^*)$ et $\exists \gamma_i \in \{\Box\}^*\ \gamma_i\mu_i = \alpha_i$.

La machine $\mathcal{M}'$ simule le fonctionnement de $\mathcal{M}$ de la façon suivante. Tout d'abord, afin de simuler la configuration initiale de $\mathcal{M}$, elle recopie le mot $\%_1\ \dot{\Box}\ \%_2\ \ldots\ \%_k\ \dot{\Box}\ \%_{k+1}$ sur sa bande de travail. Soit δ_{copie} l'ensemble des transitions de $\mathcal{M}'$ qui permettent de réaliser ces opérations et Q_{copie} l'ensemble des états qui interviennent dans δ_{copie}. On suppose qu'après avoir effectué cette recopie, $\mathcal{M}'$ passe à l'état q_0.

Tout travail de $\mathcal{M}$ consiste à écrire un caractère Y_i à la place d'un caractère X_i sur chaque bande de travail i et à faire avancer ou pas la tête de i vers la gauche ou vers la droite. Cela est simulé par $\mathcal{M}'$ de la façon suivante. Si $\mathcal{M}$ ne modifie pas la position de la tête de i, alors $\mathcal{M}'$ écrit juste $\dot{Y}_i$ à la place de $\dot{X}_i$. Dans le cas où $\mathcal{M}$ fait avancer la tête de i vers la gauche (resp. vers la droite), $\mathcal{M}'$ écrit Y_i à la place de $\dot{X}_i$ et, si Z est le caractère inscrit à gauche (resp. à droite) de Y_i, elle remplace Z par $\dot{Z}$. La machine $\mathcal{M}'$ effectue toutes ces opérations sur la i^{e} portion de sa bande de travail (*i.e.* la partie de cette bande correspondant à la i^{e} bande de $\mathcal{M}$) en passant dans l'état $q_{t,i}$ (où t est la transition de $\mathcal{M}$ qui provoque toutes ces opérations effectuées par $\mathcal{M}'$). De plus, $\mathcal{M}'$ doit être capable de déplacer la tête de sa bande de travail depuis la i^{e} portion de la bande jusqu'à la suivante vers la droite (si cette suivante existe). C'est en passant dans l'état $m_{t,i}$ que $\mathcal{M}'$ effectue ces déplacements. Enfin, lorsque la k^{e} portion de la bande a subi tous ces traitements, $\mathcal{M}'$ passe à l'état $m_{t,k}$ et déplace la tête de sa bande de travail jusqu'à la première portion.

À chaque fois que $\mathcal{M}$ place la tête de la bande i avant le début (resp. après la fin) de l'inscription, $\mathcal{M}'$ doit insérer $\dot{\Box}$ à droite de $\%_i$ (resp. à gauche de $\%_{i+1}$). Afin de réaliser cela, elle décale vers la gauche (resp. la droite) la partie de l'inscription de sa bande de travail qui va de $\%_1$ à $\%_i$ (resp. de $\%_{i+1}$ à $\%_{k+1}$) puis écrit $\dot{\Box}$ dans la bonne cellule. Ces opérations sont effectuées grâce à un ensemble de transitions appelé $\delta_{\text{décalage}}$ et commencent à l'état $l_{t,i} \in Q_{\text{décalage}}$ (resp. $r_{t,i+1} \in Q_{\text{décalage}}$) où $Q_{\text{décalage}}$ est l'ensemble des états qui interviennent dans $\delta_{\text{décalage}}$. On suppose qu'après le décalage de la i^{e} portion de sa bande et l'écriture de $\dot{\Box}$, $\mathcal{M}'$ passe à l'état $m_{t,i}$.

Finalement, on pose $\mathcal{M}' = (Q', \Sigma, \Gamma', \delta_{\text{copie}} \cup \delta_{\text{décalage}} \cup \delta', q_0{}')$ où :

- $Q' = Q \cup \{q_{t,i} \mid t \in \delta \text{ et } i \in [k]\} \cup \{m_{t,i} \mid t \in \delta \text{ et } i \in [k]\} \cup Q_{\text{copie}} \cup Q_{\text{décalage}}$,
- $\Gamma' = \Gamma_1 \cup \cdots \cup \Gamma_k \cup \{\%_1, \ldots, \%_{k+1}\} \cup \dot{\Gamma}_1 \cup \cdots \cup \dot{\Gamma}_k$ et
- δ' est défini par : $t = (q, c, X_1, \ldots, X_k, Y_1, \blacklozenge_1, Y_2, \blacklozenge_2, \ldots, Y_k, \blacklozenge_k, q') \in \delta$ si et seulement si l'ensemble suivant est une partie de δ' :

$$\big\{(q, c, \dot{X}_1, Y_1, \blacklozenge_1, q_{t,1})\big\} \;\bigcup\; \big\{(q_{t,i}, \varepsilon, X, \dot{X}, \blacktriangleright, m_{t,i}) \bigm| i \in [k-1],\ X \in \Gamma_i\big\} \;\bigcup$$

$$\big\{(q_{t,i}, \varepsilon, \%_i, \%_i, \blacksquare, l_{t,i}) \bigm| i \in [k]\big\} \;\bigcup\; \big\{(q_{t,i}, \varepsilon, \%_{i+i}, \%_{i+1}, \blacksquare, r_{t,i+1}) \bigm| i \in [k]\big\} \;\bigcup$$

$$\big\{(q_{t,k}, \varepsilon, X, \dot{X}, \blacksquare, m_{t,k}) \bigm| X \in \Gamma_k\big\} \;\bigcup$$

$$\big\{(m_{t,i}, \varepsilon, X, X, \blacktriangleright, m_{t,i}) \bigm| i \in [k-1],\ X \in \Gamma_i \cup \{\%_{i+1}\} \cup \Gamma_{i+1}\big\} \;\bigcup$$

$$\big\{(m_{t,i}, \varepsilon, \dot{X}_{i+1}, Y_{i+1}, \blacklozenge_{i+1}, q_{t,i+1}) \bigm| i \in [k-1]\big\} \;\bigcup$$

$$\big\{(m_{t,k}, \varepsilon, X, X, \blacktriangleleft, m_{t,k}) \bigm| X \in \textstyle\bigcup_{i\in[k]} \Gamma_i \cup \{\%_2, \ldots, \%_k\} \cup \bigcup_{i\in[k]\smallsetminus\{1\}} \dot{\Gamma}_i\big\} \;\bigcup$$

$$\big\{(m_{t,k}, \varepsilon, \dot{X}, \dot{X}, \blacksquare, q') \bigm| X \in \Gamma_1\big\}\ .$$

Par conséquent, les configurations internes de $\mathcal{M}'$ sont des éléments de

$$\%_1.{\Gamma_1}^*.Q'.{\Gamma_1}^*.\dot{\Gamma}_1.{\Gamma_1}^*.\%_2.{\Gamma_2}^*.\dot{\Gamma}_2.{\Gamma_2}^*.\%_3\ \ldots \%_k{\Gamma_k}^*.\dot{\Gamma}_k.{\Gamma_k}^*.\%_{k+1}\ \cup\ \ldots$$

(la tête de la bande de travail pouvant être placée n'importe où, les points de suspension signifient « la même chose avec Q' n'importe où ailleurs »).

Notons que le nombre d'états et de transitions de $\mathcal{M}'$ sont respectivement en $O\big(k.|\delta|\big)$ et en $O\big(|\delta|.\sum_{i\in[k]}|\Gamma_i|\big)$.

Vu la définition de $\mathcal{M}'$, la proposition suivante est immédiate.

Proposition 5.1 *Le graphe de la machine à k bandes de travail $\mathcal{M}$ et celui de la machine à une seule bande travail $\mathcal{M}'$ construite à partir de $\mathcal{M}$ sont ε-équivalents.*

Le cas où $\mathcal{M}$ est linéairement bornée a été étudié dans [KP99]. Dans cet article, nous remarquons qu'en partant d'une telle machine, la construction définie dans cette partie produit une machine $\mathcal{M}'$ qui est également linéairement bornée. En effet, le nombre de cellules que $\mathcal{M}'$ utilise sur sa bande de travail pour traiter un mot d'entrée de taille n est $(k+1)+\sum_{i\in[k]} S_i(n)$ où $k+1$ représente le nombre de cellules contenant un délimiteur et les S_i sont les k fonctions linéaires associées à $\mathcal{M}$.

3 Fermeture par produit synchronisé

La classe $\mathcal{G}[\text{TM}]$ est fermée, à isomorphisme près, par produit synchronisé. En effet, soit n machines de Turing $\mathcal{M}_i = (Q_i, \Sigma_i, \Gamma_{i,1}, \ldots, \Gamma_{i,k_i}, \delta_i, q_{0i})$, $i \in [n]$, et une contrainte de synchronisation $\mathscr{C} \subseteq \prod_{i\in[n]} \widetilde{\Sigma_i}$.

Soit $\mathcal{M}$ la machine de Turing à $\sum_{i\in[n]} k_i$ bandes de travail définie ainsi : l'ensemble des états est $\prod_{i\in[n]} Q_i$, l'alphabet d'entrée est $\mathscr{C}$, les alphabets des bandes de travail sont tous les $\Gamma_{i,j}$ pour $i \in [n]$ et $j \in [k_i]$, l'état initial est $(q_{01}, \dots, q_{0n})$, et l'ensemble des transitions contient tous les uplets

$$\Big(q, a, (X_{1,j})_{j\in[k_1]}\ , \dots, (X_{n,j})_{j\in[k_n]}\ , (Y_{1,j}, \blacklozenge_{1,j})_{j\in[k_1]}\ , \dots, (Y_{n,j}, \blacklozenge_{n,j})_{j\in[k_n]}\ , q'\Big)$$

tels que $a \in \mathscr{C}$ et, pour tout $i \in [n]$,

$$\big(\pi_i\big(q\big), \pi_i\big(a\big), (X_{i,j})_{j\in[k_i]}\ , (Y_{i,j}, \blacklozenge_{i,j})_{j\in[k_i]}\ , \pi_i\big(q'\big)\big) \in \delta_i\ .$$

Ici, pour tout $i \in [n]$, $(X_{i,j})_{j\in[k_i]}$ représente la suite $X_{i,1}\ , \dots, X_{i,k_i}$ et $(Y_{i,j}, \blacklozenge_{i,j})_{j\in[k_i]}$ la suite $Y_{i,1}\ , \blacklozenge_{i,1}\ , \dots, Y_{i,k_i}\ , \blacklozenge_{i,k_i}$.

Proposition 5.2 *Le graphe de la machine $\mathcal{M}$ construite à partir des machines $\mathcal{M}_i$ qu'on s'est données au départ est isomorphe au produit $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{M}_i)$.*

Preuve. Soit ϕ la bijection

$$\prod_{i\in[n]} \Big(\prod_{j\in[k_i]} {\Gamma_{i,j}}^*.Q_i.{\Gamma_{i,j}}^*\Big) \longrightarrow \prod_{i\in[n],j\in[k_i]} {\Gamma_{i,j}}^*.\Big(\prod_{l\in[n]} Q_l\Big).{\Gamma_{i,j}}^* \quad \text{telle que}$$

$$\phi\Big((\mu_{1,1}q_1\nu_{1,1}\ , \dots, \mu_{1,k_1}q_1\nu_{1,k_1})\ , \dots, (\mu_{n,1}q_n\nu_{n,1}\ , \dots, \mu_{n,k_n}q_n\nu_{n,k_n})\Big)$$
$$= \big(\mu_{1,1}(q_1, \dots, q_n)\nu_{1,1}\ , \dots, \mu_{n,k_n}(q_1, \dots, q_n)\nu_{n,k_n}\big)\ .$$

Soit ι la configuration initiale de $\mathcal{M}$ et $\iota_1, \dots, \iota_n$ la configuration initiale de $\mathcal{M}_1, \dots, \mathcal{M}_n$ respectivement.

Le graphe $\mathcal{G}(\mathcal{M})$ a pour racine distinguée ι et $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{M}_i)$ a pour racine distinguée $(\iota_1, \dots, \iota_n)$ (nous rappelons que dans le cas du produit synchronisé de graphes ayant une racine distinguée, nous utilisons une variante du produit synchronisé usuel, voir le paragraphe 5 du chapitre 2). Les racines $(\iota_1, \dots, \iota_n)$ et ι vérifient $\phi\big(\iota_1, \dots, \iota_n\big) = \iota$. De plus, étant donnée la définition des transitions de $\mathcal{M}$, il est clair que $d \xrightarrow{a} d'$ est un arc de $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{M}_i)$ si et seulement si $\phi(d) \xrightarrow{a} \phi(d')$ est un arc de $\mathcal{G}(\mathcal{M})$. □

Notons que la construction décrite dans la partie précédente permet d'associer à $\mathcal{M}$ une machine à une seule bande de travail dont le graphe est ε-équivalent à celui de $\mathcal{M}$. On en déduit donc que la classe des graphes des machines de Turing à une seule bande de travail est fermée, à ε-équivalence près, par produit synchronisé.

Enfin, comme nous l'avons noté dans [KP99], si toutes les machines $\mathcal{M}_i$ sont linéairement bornées, la machine $\mathcal{M}$ construite ci-dessus est elle aussi linéairement bornée. Par conséquent, la classe $\mathcal{G}$[Lin-TM] des machines de Turing linéairement bornées est fermée, à isomorphisme près, par produit synchronisé.

Nous résumons ces résultats de fermeture dans le théorème suivant.

Théorème 5.3 *Les classes $\mathcal{G}$[TM] et $\mathcal{G}$[Lin-TM] sont fermées, à isomorphisme près, par produit synchronisé. La restriction de ces classes aux graphes des machines à une seule bande de travail est fermée, à ε-équivalence près, par produit synchronisé.*

4 Liens avec les graphes des spécifications de Thue

Dans cette partie, nous proposons deux constructions. L'une permet d'associer à n'importe quelle spécification de Thue une machine de Turing dont le graphe est ε-équivalent à celui de la spécification. L'autre permet d'associer à une machine de Turing vérifiant des conditions bien précises une spécification de Thue dont le graphe est ε-équivalent à celui de la machine. Pour chacune de ces deux constructions, nous fournissons un exemple qui devrait faciliter la compréhension du lecteur.

4.1 Machine de Turing associée à une spécification de Thue

Soit $\mathcal{T} = (\Omega, \Delta, \mathcal{S}, R, u)$ une spécification de Thue. Nous construisons une machine de Turing $\mathcal{M}$ qui « simule » $\mathcal{T}$, plus précisément dont le graphe est ε-équivalent à celui de $\mathcal{T}$.

La machine $\mathcal{M}$ est constituée de deux bandes de travail appelées B_1 et B_2 et fonctionne ainsi. Les premières opérations qu'elle effectue à partir de sa configuration initiale (q_0, q_0) consistent à écrire le mot u sur B_1 et B_2 et à se placer dans la configuration (uq_0', uq_0').

On pourrait penser à concevoir $\mathcal{M}$ de telle sorte qu'elle lise ensuite un caractère sur sa bande d'entrée : si elle lit a, elle pourrait alors écrire cette lettre à la fin de l'inscription u de B_1 et se mettre à réduire le mot ua en utilisant les règles de $\mathcal{S}$. Une telle conception pose cependant un problème si aucune réécriture de ua n'aboutit dans R. En effet, dans ce cas, aucun arc étiqueté par a ne sort du sommet u dans le graphe de $\mathcal{T}$. Par contre, comme $\mathcal{M}$ a lu a, il y a dans son graphe un arc étiqueté par a ayant pour origine sa configuration initiale. Il n'y a donc pas d'ε-équivalence dans ce cas-là.

Il est possible d'éviter à $\mathcal{M}$ d'avoir à lire sur sa bande d'entrée avant de réduire si l'on remarque le fait suivant. Lorsqu'on réécrit un mot wa tel que $w \in R$ et $a \in \Delta$, la première règle qu'on utilise s'applique obligatoirement à un suffixe la de wa car w est irréductible par $\mathcal{S}$. Cette première règle a donc la forme $la \to r$ et, en supposant que $w = w'l$, l'appliquer à wa revient à transformer $w'l$ en $w'r$.

On conçoit donc plutôt $\mathcal{M}$ de telle sorte que son comportement implémente l'algorithme décrit à la figure 5.1. Il est important de noter qu'alors $\mathcal{M}$ ne lit une lettre a sur sa bande d'entrée que dans deux cas bien précis (on appelle w l'inscription de la bande B_1 au départ de la procédure Simuler) :

1. quand wa est dans R, c'est-à-dire quand dans $\mathcal{G}(\mathcal{T})$ il y a un arc étiqueté par a allant de w à wa ;
2. quand une réécriture de wa commençant par une règle $la \to r \in \mathcal{S}$ a abouti au mot $w' \in R$, c'est-à-dire quand dans $\mathcal{G}(\mathcal{T})$ il y a un arc étiqueté par a allant de w à w'.

Il faut toutefois signaler que la boucle **tant que** ... **faire** de la procédure Simuler ne termine pas dans le cas où le mot que l'on réécrit n'a aucune forme normale par rapport au système de réécriture $\mathcal{S}$. Par conséquent, pour que la machine $\mathcal{M}$ ne reste pas bloquée en réécrivant un tel mot, on la conçoit de telle sorte qu'elle implémente l'algorithme de la façon suivante : à partir du mot w de R inscrit sur sa bande de travail B_1, elle peut effectuer, pour tout a de Σ, une transition qui lance les opérations de la boucle **pour chaque** $a \in \Delta$ **faire** correspondant à ce a. La machine $\mathcal{M}$ est donc complètement indéterministe.

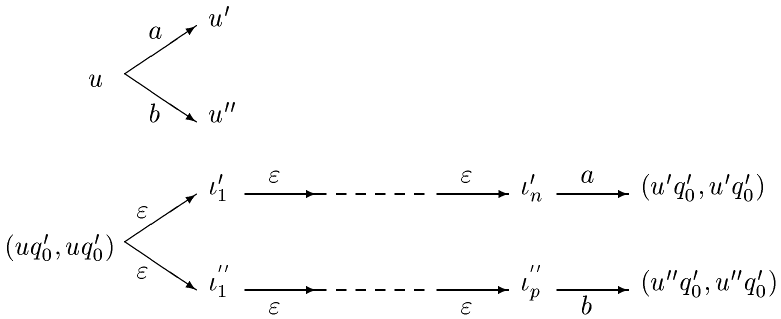
```
procédure M
    passer en configuration (uq'_0, uq'_0) ;
    Simuler ;

procédure Simuler
    w := inscription(B_1) ;
    pour chaque a ∈ Δ faire
        si wa ∈ R alors Terminer[a, wa] ;
        sinon
            pour chaque la → r ∈ S tq. w = w'l faire
                transformer l'inscription w'l de B_1 en w'r ;
                tant que inscription(B_1) ∉ R faire
                    prendre une règle l' → r' de S ;
                    appliquer l' → r' à l'inscription de B_1 ;
                Terminer[a, inscription(B_1)] ;

procédure Terminer[a ∈ Δ, w ∈ R]
    lire a sur la bande d'entrée ;
    passer en configuration (wq'_0, wq'_0) ;
    Simuler ;
```

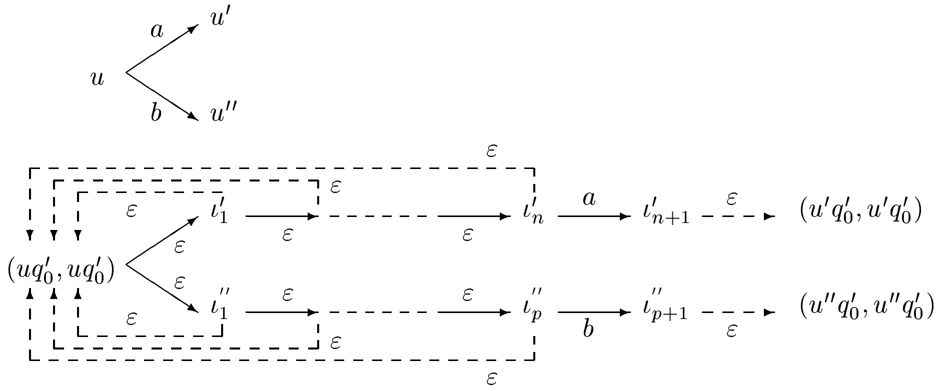
FIG. 5.1 – *Fonctionnement de* $\mathcal{M}$

Cependant, le graphe de la machine $\mathcal{M}$ construite jusqu'ici n'est pas toujours ε-équivalent à celui de $\mathcal{T}$. En effet, considérons l'exemple ci-dessous. Nous avons représenté, en haut, le graphe d'une spécification de Thue et, en bas, une partie du graphe de la machine de Turing correspondant. Il est facile de vérifier que ces deux graphes ne sont pas ε-équivalents. En effet, la seule relation d'ε-équivalence possible serait telle que $u \leftrightsquigarrow (uq'_0, uq'_0)$, $u' \leftrightsquigarrow (u'q'_0, u'q'_0)$, $u'' \leftrightsquigarrow (u''q'_0, u''q'_0)$, pour tout $i \in [n]$, $\iota'_i \leftrightsquigarrow u$ et pour tout $i \in [p]$, $\iota''_i \leftrightsquigarrow u$. Or, on a $u \xrightarrow{b} u''$, $u \leftrightsquigarrow \iota'_1$ et il n'existe aucun sommet ι tel que $u'' \leftrightsquigarrow \iota$ et $\iota'_1 \overset{b}{\dashrightarrow} \iota$.



Pour que ce problème ne se pose pas, *i.e.* pour que l'on puisse trouver un sommet ι convenable, on conçoit $\mathcal{M}$ de telle sorte qu'à chaque étape d'une réduction elle puisse revenir à la configuration qu'elle avait avant de commencer à réduire. On utilise pour cela la seconde bande de travail qui mémorise le mot qui est sur B_1 avant la réduction. Remarquons que si la réduction d'un mot

wa termine par $w' \in R$, $\mathcal{M}$ doit inscrire w' à la place de w sur B_2. L'exemple précédent devient alors :



où les transitions qui démarrent à partir de ι'_{n+1} (resp. de ι''_{p+1}) correspondent à la copie de u' (resp. u'') sur B_2. Il est facile de voir que ces deux graphes sont ε-équivalents.

Enfin, à chaque fois que $\mathcal{M}$ réécrit un mot w_1lw_2 en utilisant une règle $l \to r$, elle inscrit r à la place de l. Si $l = l_1l_2$, avec $|l_1| = |r|$ et $|l_2| \neq \varepsilon$, elle inscrit r à la place de l_1, puis efface l_2 et finalement décale vers la gauche la portion de la bande correspondant à w_2. Si $r = r_1r_2$, avec $|r_1| = |l|$ et $|r_2| \neq \varepsilon$, $\mathcal{M}$ inscrit r_1 sur l puis décale à droite la portion de B_1 correspondant à w_2 et enfin écrit r_2 à la fin de r_1. Après que ces opérations ont été réalisées, la machine vérifie si le mot obtenu par application de la règle $l \to r$ est dans R.

Voici une définition plus formelle de $\mathcal{M}$. Soit $\mathcal{A} = (Q_{\mathcal{A}}, \Omega, \delta_{\mathcal{A}}, p_0, F)$ un automate fini, déterministe, complet et sans ε-transitions qui reconnaît R. On pose

$$\mathcal{M} = (Q_{\mathcal{M}}, \Delta, \Omega \cup \Delta \cup \{\square\}, \Omega \cup \{\square\}, \delta_{\mathcal{M}}, q_0)\ .$$

L'ensemble des états $Q_{\mathcal{M}}$ est

$$\begin{aligned} Q_{\mathcal{M}} = \quad & \{q_0, q'_0\} \ \cup\ Q_u \ \cup\ Q_{\text{retour}} \ \cup\ Q_{\text{remplacer}} \ \cup\ Q_{\text{décaler}} \ \cup \\ & \big\{[a], [a, \in] \mid a \in \Delta\big\} \ \cup\ \big\{[a, p] \mid a \in \Delta,\ p \in Q_{\mathcal{A}}\big\} \cup \\ & \big\{[\overline{a}, \in] \mid a \in \Omega \cap \Delta\big\} \ \cup\ \big\{[[\overline{a}, p] \mid a \in \Omega \cap \Delta,\ p \in Q_{\mathcal{A}}\big\} \cup \\ & \big\{[a, l \to r \mid w] \mid a \in \Delta,\ l \to r \in \mathcal{S},\ w \in \mathcal{S}\mathrm{uff}^+(l)\big\} \cup \\ & \big\{[a \mid w, h] \mid a \in \Delta,\ \exists l \to r \in \mathcal{S},\ w \in \mathcal{S}\mathrm{uff}(l) \text{ et } h \in \mathcal{S}\mathrm{uff}(r)\big\} \end{aligned}$$

où Q_u, Q_{retour}, $Q_{\text{remplacer}}$ et $Q_{\text{décaler}}$ sont les ensembles d'états qui interviennent dans δ_u, δ_{retour}, $\delta_{\text{remplacer}}$ et $\delta_{\text{décaler}}$ (voir ci-dessous).

L'ensemble des transitions est

$$\delta_{\mathcal{M}} = \delta_u \cup \delta_{\text{retour}} \cup \delta_{\text{remplacer}} \cup \delta_{\text{décaler}} \cup \delta_{\text{départ}} \cup \delta_{\text{vérifier-}wa} \cup \delta_{\text{réécrire}} \cup \delta_{\text{vérifier}} \cup \delta_{\text{chercher}}$$

où δ_u, δ_{retour}, $\delta_{\text{remplacer}}$ et $\delta_{\text{décaler}}$ sont les ensembles d'ε-transitions qui permettent à la machine d'effectuer les opérations décrites plus haut : δ_u est l'ensemble d'ε-transitions qui permettent à

$\mathcal{M}$ d'inscrire le mot u sur ses bandes de travail (lorsque $\mathcal{M}$ a inscrit u, elle se place dans la configuration (uq_0', uq_0')), δ_{retour} est l'ensemble d'ε-transitions qui permettent à $\mathcal{M}$ de revenir à la configuration qu'elle avait avant de commencer à réécrire un mot, $\delta_{\text{remplacer}}$ est l'ensemble d'ε-transitions qui permettent à $\mathcal{M}$ d'inscrire sur B_2 le contenu de B_1 lorsque celui-ci est le résultat d'une réécriture qui a abouti dans R et $\delta_{\text{décaler}}$ est l'ensemble d'ε-transitions qui permettent à $\mathcal{M}$ de décaler une partie de l'inscription de B_1 lors de l'application d'une règle.

Nous supposons que les ensembles de transitions δ_u et $\delta_{\text{remplacer}}$ sont déterministes, c'est-à-dire que pour toute configuration $(\mu_1 q \nu_1, \mu_2 q \nu_2)$ telle que $q \in Q_u$ (resp. $q \in Q_{\text{remplacer}}$), s'il existe dans δ une transition de la forme $(q, \varepsilon, X_1, X_2, Y_1, \blacklozenge_1, Y_2, \blacklozenge_2, p)$ où

- X_1 est la première lettre de ν_1 si ce mot n'est pas vide et $X_1 = \square$ sinon et
- X_2 est la première lettre de ν_2 si ce mot n'est pas vide et $X_2 = \square$ sinon,

toute transition de δ de la forme $(q', \varepsilon, X_1', X_2', Y_1', \blacklozenge_1', Y_2', \blacklozenge_2', p')$ avec $q' \in Q_u$ (resp. $q' \in Q_{\text{remplacer}}$) est telle que :

$$(Y_1', \blacklozenge_1', Y_2', \blacklozenge_2', p') \neq (Y_1, \blacklozenge_1, Y_2, \blacklozenge_2, p) \Rightarrow \; (q', \varepsilon, X_1', X_2') \neq (q, \varepsilon, X_1, X_2).$$

Les ensembles $\delta_{\text{départ}}$, $\delta_{\text{vérifier-}wa}$, $\delta_{\text{réécrire}}$, $\delta_{\text{vérifier}}$ et δ_{chercher} sont quant à eux définis comme suit. Les transitions de $\delta_{\text{départ}}$ lancent la première boucle **pour chaque** ... **faire** de la procédure Simuler intervenant dans l'algorithme décrit à la figure 5.1 :

$$\begin{aligned}\delta_{\text{départ}} \;\; = \;\; & \Big\{ \Big(q_0', \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, \big[\overline{a}, \in\,\big] \Big) \;\Big|\; a \in \Omega \cap \Delta \Big\} \bigcup \\ & \Big\{ \Big(q_0', \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, \big[a, q_0'\big] \Big) \;\Big|\; a \in \Delta,\; a \notin \Omega \Big\}.\end{aligned}$$

Soit w l'inscription de B_1. Il est clair que si un élément a de Δ n'est pas dans Ω, wa ne peut appartenir à R car $R \subseteq \Omega^*$. Dans ce cas, la machine $\mathcal{M}$ tente directement de réécrire wa à partir de l'état $[a, q_0']$. Sinon, à partir de l'état $\big[\overline{a}, \in\,\big]$, la machine vérifie si $wa \in R$ au moyen des transitions de l'ensemble $\delta_{\text{vérifier-}wa}$:

$$\begin{aligned}\delta_{\text{vérifier-}wa} = \;\; & \Big\{ \Big(\big[\overline{a}, \in\,\big], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, \big[\overline{a}, \in\,\big] \Big) \;\Big|\; a \in \Omega \cap \Delta,\; x \in \Omega \Big\} \bigcup \\ & \Big\{ \Big(\big[\overline{a}, \in\,\big], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacksquare, \big[\overline{a}, p_0\big] \Big) \;\Big|\; a \in \Omega \cap \Delta \Big\} \bigcup \\ & \Big\{ \Big(\big[\overline{a}, p\big], \varepsilon, x, \square, x, \blacktriangleright, \square, \blacksquare, \big[\overline{a}, p'\big] \Big) \;\Big|\; a \in \Omega \cap \Delta,\; x \in \Omega,\; p, p' \in Q_{\mathcal{A}},\; (p, x, p') \in \delta_{\mathcal{A}} \Big\} \bigcup \\ & \Big\{ \Big(\big[\overline{a}, p\big], a, \square, \square, a, \blacktriangleright, a, \blacktriangleright, q_0' \Big) \;\Big|\; a \in \Omega \cap \Delta,\; p \in Q_{\mathcal{A}},\; \exists f \in F \text{ tq. } (p, a, f) \in \delta_{\mathcal{A}} \Big\} \bigcup \\ & \Big\{ \Big(\big[\overline{a}, p\big], \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, \big[a, q_0'\big] \Big) \;\Big|\; \begin{array}{l} a \in \Omega \cap \Delta,\; p \in Q_{\mathcal{A}},\; \forall p' \in Q_{\mathcal{A}}, \\ (p, a, p') \in \delta_{\mathcal{A}} \Rightarrow p' \notin F \end{array} \Big\}.\end{aligned}$$

Pour un a donné, si le mot wa est un élément de R, la machine $\mathcal{M}$ peut lire a sur sa bande d'entrée, inscrire a à la fin de l'inscription de ses deux bandes de travail et recommencer en revenant à l'état q_0'. Sinon (*i.e.* si wa est réductible ou si $wa \in \mathrm{Irr}(\mathcal{S}) \setminus R$), $\mathcal{M}$ se branche obligatoirement sur l'état $[a, q_0']$ car on a supposé que l'automate $\mathcal{A}$ est déterministe et complet. À partir de cet état, $\mathcal{M}$ tente alors de réécrire wa au moyen des règles de $\mathcal{S}$. Ce sont les transitions de l'ensemble $\delta_{\text{réécrire}}$ qui lui permettent de réaliser cela :

$$
\begin{aligned}
\delta_{\text{réécrire}} = \quad & \Big\{ \Big([a, q_0'], \varepsilon, x, \square, x, \blacktriangleright, \square, \blacksquare, [a \mid \varepsilon, r] \Big) \;\Big|\; a \to r \in \mathcal{S},\ a \in \Delta,\ x \in \Omega \cup \{\square\} \Big\} \cup \\
& \Big\{ \Big([a, q_0'], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, [a, l \to r \mid xa] \Big) \;\Big|\; \begin{array}{l} a \in \Delta,\ l \to r \in \mathcal{S}, \\ x \in \Omega,\ xa \in \mathrm{Suff}(l) \end{array} \Big\} \cup \\
& \Big\{ \Big([a, l \to r \mid g], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, [a, l \to r \mid xg] \Big) \;\Big|\; \begin{array}{l} a \in \Delta,\ l \to r \in \mathcal{S},\ x \in \Omega \cup \Delta, \\ g \in \mathrm{Suff}^+(l),\ xg \in \mathrm{Suff}(l) \end{array} \Big\} \cup \\
& \Big\{ \Big([a, l \to r \mid l], \varepsilon, z, \square, z, \blacktriangleright, \square, \blacksquare, [a \mid l, r] \Big) \;\Big|\; a \in \Delta,\ l \to r \in \mathcal{S},\ z \in \Omega \cup \Delta \cup \{\square\} \Big\} \cup \\
& \left\{ \Big([a \mid g, h], \varepsilon, x, \square, x', \blacktriangleright, \square, \blacksquare, [a \mid g', h'] \Big) \;\middle|\; \begin{array}{l} a \in \Delta,\ x, x' \in \Omega \cup \Delta,\ \exists l \to r \in \mathcal{S}, \\ g \in \mathrm{Suff}(l),\ h \in \mathrm{Suff}(r), \\ g = xg',\ h = x'h' \end{array} \right\} \cup \\
& \Big\{ \Big([a \mid g, \varepsilon], \varepsilon, x, \square, x, \blacksquare, \square, \blacksquare, [a, \blacktriangleleft \mid g] \Big) \;\Big|\; \begin{array}{l} a \in \Delta,\ \exists l \to r \in \mathcal{S},\ g \in \mathrm{Suff}^+(l), \\ x = g(1) \end{array} \Big\} \cup \\
& \Big\{ \Big([a \mid \varepsilon, h], \varepsilon, z, \square, z, \blacksquare, \square, \blacksquare, [a, \blacktriangleright \mid h] \Big) \;\Big|\; \begin{array}{l} a \in \Delta,\ z \in \Omega \cup \Delta \cup \{\square\}, \\ \exists l \to r \in \mathcal{S},\ h \in \mathrm{Suff}^+(r) \end{array} \Big\} \cup \\
& \Big\{ \Big([a \mid \varepsilon, \varepsilon], \varepsilon, z, \square, z, \blacktriangleleft, \square, \blacksquare, [a, \in] \Big) \;\Big|\; a \in \Delta,\ z \in \Omega \cup \Delta \cup \{\square\} \Big\} .
\end{aligned}
$$

Les quatre premiers ensembles de cette union contiennent les transitions qui effectuent la recherche d'un membre gauche de règle. Les quatre autres contiennent les transitions qui effectuent l'application d'une règle. Les états $[a, \blacktriangleleft \mid g]$ et $[a, \blacktriangleright \mid g]$ sont ceux qui lancent les transitions de l'ensemble $\delta_{\text{décaler}}$ qui sont celles qui permettent à la machine de décaler une portion de l'inscription de B_1 au moment de l'application d'une règle. Les états $[a, \blacktriangleleft \mid g]$ interviennent dans le cas où il faut effectuer un décalage vers la gauche et les états $[a, \blacktriangleright \mid g]$ dans le cas d'un décalage vers la droite. On suppose qu'après tout décalage débutant dans un état $[a, \blacktriangleleft \mid g]$ ou $[a, \blacktriangleright \mid g]$, $\mathcal{M}$ passe dans l'état $[a, \in]$.

Lorsque $\mathcal{M}$ a appliqué une règle, elle doit vérifier si le mot obtenu est dans R. Ce sont les transitions de $\delta_{\text{vérifier}}$ qui lui permettent de réaliser cela :

$$\begin{aligned}
\delta_{\text{vérifier}} = \; & \Big\{\Big(\big[a,\in\big], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, \big[a,\in\big]\Big) \;\Big|\; a\in\Delta,\ x\in\Omega\Big\} \;\bigcup \\
& \Big\{\Big(\big[a,\in\big], \varepsilon, x, \square, x, \blacksquare, \square, \blacksquare, \big[a\big]\Big) \;\Big|\; a\in\Delta,\ x\notin\Omega\Big\} \;\bigcup \\
& \Big\{\Big(\big[a,\in\big], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacksquare, \big[a,p_0\big]\Big) \;\Big|\; a\in\Delta\Big\} \;\bigcup \\
& \Big\{\Big(\big[a,p\big], \varepsilon, x, \square, x, \blacktriangleright, \square, \blacksquare, \big[a,p'\big]\Big) \;\Big|\; a\in\Delta,\ x\in\Omega,\ p,p'\in Q_{\mathcal{A}},\ (p,x,p')\in\delta_{\mathcal{A}}\Big\} \;\bigcup \\
& \Big\{\Big(\big[a,p\big], a, \square, \square, \square, \blacktriangleleft, \square, \blacktriangleleft, \big[\text{remplacer}\big]\Big) \;\Big|\; p\in F,\ a\in\Delta\Big\} \;\bigcup \\
& \Big\{\Big(\big[a,p\big], \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, \big[a\big]\Big) \;\Big|\; p\in Q_{\mathcal{A}}\setminus F,\ a\in\Delta\Big\}\;.
\end{aligned}$$

Si le mot w' obtenu par application d'une règle au mot wa est un élément de R, la machine lit a sur sa bande d'entrée et remplace l'inscription de B_2 par celle de B_1 (c'est-à-dire w'). Pour effectuer ce remplacement, elle passe dans l'état $\big[\text{remplacer}\big]$ qui lance les transitions de $\delta_{\text{remplacer}}$. Après tout remplacement de ce type, la machine $\mathcal{M}$ passe dans la configuration $(w'q_0', w'q_0')$. Si le mot w' n'est pas un élément de R, la machine continue à réécrire en cherchant un membre gauche de règle dans l'inscription de B_1 :

$$\begin{aligned}
\delta_{\text{chercher}} = \; & \Big\{\Big(\big[a\big], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, \big[a\big]\Big) \;\Big|\; a\in\Delta,\ x\in\Omega\cup\Delta\Big\} \;\bigcup \\
& \Big\{\Big(\big[a\big], \varepsilon, x, \square, x, \blacktriangleright, \square, \blacksquare, \big[a\big]\Big) \;\Big|\; a\in\Delta,\ x\in\Omega\cup\Delta\Big\} \;\bigcup \\
& \Big\{\Big(\big[a\big], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, \big[a, l\to r \mid x\big]\Big) \;\Big|\; l\to r\in\mathcal{S},\ a\in\Delta,\ x\in\mathrm{Suff}(l)\Big\}\;.
\end{aligned}$$

Nous insistons sur le fait qu'à partir de tout état intervenant dans $\delta_{\text{vérifier-}wa}$, $\delta_{\text{réécrire}}$, $\delta_{\text{décaler}}$, $\delta_{\text{vérifier}}$ et δ_{chercher}, la machine a la possibilité d'effectuer des transitions de δ_{retour} qui la ramènent à la configuration (wq_0', wq_0'). Cela lui permet d'engendrer un graphe qui est ε-équivalent à celui de $\mathcal{T}$ comme cela est expliqué plus haut.

Les lemmes suivants sont des conséquences directes de la définition de $\mathcal{M}$.

Lemme 5.4 *Chaque sommet $(\mu q\nu, \mu' q\nu')$ de $\mathcal{G}(\mathcal{M})$ vérifie l'un des points suivants.*

1. *Ou bien $q = q_0'$, $\mu = \mu'$, $\mu\in R$ et $\nu = \nu' = \varepsilon$.*
2. *Ou bien $q\in Q_{\text{remplacer}}$, $\mu\nu\in R$ et il existe une unique suite $v_1, \ldots, v_m$ de sommets de $\mathcal{G}(\mathcal{M})$ telle que*

$$(\mu q\nu, \mu' q\nu') \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} v_1 \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} \cdots \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} v_m \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} (\mu\nu q_0', \mu\nu q_0')$$

(car on a supposé que $\delta_{\text{remplacer}}$ est un ensemble déterministe de transitions). De plus, pour tout sommet v de $\mathcal{G}(\mathcal{M})$, on a $(\mu q\nu, \mu' q\nu') \overset{a}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} v$ si et seulement si

$$(\mu q\nu, \mu' q\nu') \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} v_1 \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} \cdots \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} v_m \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} (\mu\nu q_0', \mu\nu q_0') \overset{a}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} v\;.$$

3. *Ou bien $q \in Q_u$. Comme δ_u est un ensemble déterministe de transitions, ce cas est identique au précédent en remplaçant $(\mu\nu q'_0, \mu\nu q'_0)$ par (uq'_0, uq'_0).*

4. *Ou bien $q \notin Q_u \cup Q_{\text{remplacer}}$ et il existe $w \in R$ tel que $(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu q\nu, \mu' q\nu')$. De plus, pour chaque sommet (wq'_0, wq'_0) de $\mathcal{G}(\mathcal{M})$, on a :*

$$(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu q\nu, \mu' q\nu') \iff (\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (wq'_0, wq'_0) .$$

Cela est dû bien sûr aux transitions de l'ensemble δ_{retour} qui permettent à la machine de revenir en arrière à tout instant d'une réécriture.

Lemme 5.5 *Soit $w, w' \in R$ et $a \in \Delta$. Alors, on a $(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w'q'_0, w'q'_0)$ si et seulement si $w' \in (wa){\downarrow_{\mathcal{S}}}$. Par conséquent, (wq'_0, wq'_0) est un sommet de $\mathcal{G}(\mathcal{M})$ si et seulement si w est un sommet de $\mathcal{G}(\mathcal{T})$.*

Ces deux lemmes permettent de démontrer la proposition suivante.

Proposition 5.6 *Les graphes $\mathcal{G}(\mathcal{T})$ et $\mathcal{G}(\mathcal{M})$ sont ε-équivalents.*

Preuve. Soit $\leftsquigarrow\!\!\!\rightsquigarrow \subseteq R \times \left[(\Omega \cup \Delta)^*.Q.(\Omega \cup \Delta)^* \times \Omega^*.Q.\Omega^*\right]$ la relation

$$\begin{aligned} \leftrightsquigarrow \quad = \quad & \left\{(\mu\nu, (\mu q\nu, \mu' q\nu')) \mid q \in Q_{\text{remplacer}},\ (\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu\nu q'_0, \mu\nu q'_0)\right\} \cup \\ & \left\{(u, (\mu q\nu, \mu' q\nu')) \mid q \in Q_u,\ (\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (uq'_0, uq'_0)\right\} \cup \\ & \left\{(w, (\mu q\nu, wq)) \mid q \notin Q_u \cup Q_{\text{remplacer}},\ (wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu q\nu, wq)\right\} . \end{aligned}$$

On peut supposer que $q'_0 \notin Q_u \cup Q_{\text{remplacer}}$ et donc que pour tout $(wq'_0, wq'_0) \in V_{\mathcal{M}}$, $w \leftrightsquigarrow (wq'_0, wq'_0)$. Soit $V_{\mathcal{T}}$ l'ensemble des sommets de $\mathcal{G}(\mathcal{T})$ et $V_{\mathcal{M}}$ celui de $\mathcal{G}(\mathcal{M})$. Alors :

- $\mathcal{D}\text{om}(\leftrightsquigarrow) = V_{\mathcal{T}}$. En effet, la définition de $\leftrightsquigarrow$ implique que chaque $w \in \mathcal{D}\text{om}(\leftrightsquigarrow)$ est tel que (wq'_0, wq'_0) est un sommet d'un chemin dans $\mathcal{G}(\mathcal{M})$ *i.e.* est tel que $(wq'_0, wq'_0) \in V_{\mathcal{M}}$. D'après le lemme 5.5, on a donc $w \in V_{\mathcal{T}}$. Par conséquent, $\mathcal{D}\text{om}(\leftrightsquigarrow) \subseteq V_{\mathcal{T}}$.

  D'autre part, selon le lemme 5.5, si $w \in V_{\mathcal{T}}$, alors $(wq'_0, wq'_0) \in V_{\mathcal{M}}$. Par conséquent, on a $w \leftrightsquigarrow (wq'_0, wq'_0)$ *i.e.* $w \in \mathcal{D}\text{om}(\leftrightsquigarrow)$. Donc, on a également $V_{\mathcal{T}} \subseteq \mathcal{D}\text{om}(\leftrightsquigarrow)$.
- $\mathcal{R}\text{an}(\leftrightsquigarrow) = V_{\mathcal{M}}$. En effet, la définition de $\leftrightsquigarrow$ implique que tout élément de $\mathcal{R}\text{an}(\leftrightsquigarrow)$ est un sommet d'un chemin de $\mathcal{G}(\mathcal{M})$ *i.e.* que $\mathcal{R}\text{an}(\leftrightsquigarrow) \subseteq V_{\mathcal{M}}$. D'autre part, si $(\mu q\nu, \mu' q\nu') \in V_{\mathcal{M}}$, il suffit d'examiner les cas $q = q'_0$, $q \in Q_u$, $q \in Q_{\text{remplacer}}$ et $q \notin Q_u \cup Q_{\text{remplacer}}$. D'après le lemme 5.4, dans chacun de ces cas on peut trouver un mot w de R tel que $w \leftrightsquigarrow (\mu q\nu, \mu' q\nu')$. Donc, $V_{\mathcal{M}} \subseteq \mathcal{R}\text{an}(\leftrightsquigarrow)$.
- u est la racine distinguée de $\mathcal{G}(\mathcal{T})$, (q_0, q_0) est celle de $\mathcal{G}(\mathcal{M})$ et on a $u \leftrightsquigarrow (q_0, q_0)$ en supposant que $q_0 \in Q_u$.
- Soit $a \in \Delta$, w_1 et w_2 des sommets de $\mathcal{G}(\mathcal{T})$ tels que $w_1 \underset{\mathcal{G}(\mathcal{T})}{\overset{a}{\longrightarrow}} w_2$ et $(\mu q\nu, \mu' q\nu') \in V_{\mathcal{M}}$ tel que $w_1 \leftrightsquigarrow (\mu q\nu, \mu' q\nu')$. Vu la définition de $\leftrightsquigarrow$, il y a trois possibilités.
  - Ou bien $q \in Q_{\text{remplacer}}$, $\mu\nu = w_1$ et $(\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (w_1q'_0, w_1q'_0)$. Comme $w_2 \in (w_1a){\downarrow_{\mathcal{S}}}$, selon le lemme 5.5 on a également $(w_1q'_0, w_1q'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w_2q'_0, w_2q'_0)$. Par conséquent, on a

$$(\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w_2q'_0, w_2q'_0)$$

avec $w_2 \leftrightsquigarrow (w_2q'_0, w_2q'_0)$ car $(w_2q'_0, w_2q'_0)$ est un sommet de $V_{\mathcal{M}}$

- Ou bien $w_1 = u$, $q \in Q_u$ et $(\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (uq'_0, uq'_0)$. Ce cas se traite comme le précédent.
- Ou bien $(\mu q\nu, \mu' q\nu')$ est tel que $\mu' = w_1$, $\nu' = \varepsilon$ et $(w_1q'_0, w_1q'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu q\nu, \mu' q\nu')$. Dans ce cas, d'après le point 4 du lemme 5.4, on a également $(\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (w_1q'_0, w_1q'_0)$. De plus, comme $w_2 \in (w_1a){\downarrow_{\mathcal{S}}}$, d'après le lemme 5.5 on a $(w_1q'_0, w_1q'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w_2q'_0, w_2q'_0)$. Par conséquent, on a finalement $(\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w_2q'_0, w_2q'_0)$ avec $w_2 \leftrightsquigarrow (w_2q'_0, w_2q'_0)$ car $(w_2q'_0, w_2q'_0)$ est un sommet de $\mathcal{G}(\mathcal{M})$.

- Soit $a \in \Delta$, $(\mu_1q_1\nu_1, \mu'_1q_1\nu'_1)$ et $(\mu_2q_2\nu_2, \mu'_2q_2\nu'_2)$ des sommets de $\mathcal{G}(\mathcal{M})$ tels que

$$(\mu_1q_1\nu_1, \mu'_1q_1\nu'_1) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (\mu_2q_2\nu_2, \mu'_2q_2\nu'_2)$$

et w un élément de $V_{\mathcal{T}}$ tel que $w \leftrightsquigarrow (\mu_1q_1\nu_1, \mu'_1q_1\nu'_1)$.
Quelle que soit la forme de $(\mu_2q_2\nu_2, \mu'_2q_2\nu'_2)$, d'après le lemme 5.4 il existe $w' \in R$ tel que

$$(\mu_2q_2\nu_2, \mu'_2q_2\nu'_2) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (w'q'_0, w'q'_0) \quad \text{et} \quad w' \leftrightsquigarrow (\mu_2q_2\nu_2, \mu'_2q_2\nu'_2)\ .$$

De même que pour le cas précédent, il y a trois possibilités. Pour chacune d'elles, nous montrons que $(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w'q'_0, w'q'_0)$ ce qui nous permettra ensuite de conclure.

- Ou bien $q_1 \in Q_{\text{remplacer}}$, $\mu_1\nu_1 = w$ et $(\mu_1q_1\nu_1, \mu'_1q_1\nu'_1) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (wq'_0, wq'_0)$. Selon le point 2 du lemme 5.4, si $(\mu_1q_1\nu_1, \mu'_1q_1\nu'_1) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (\mu_2q_2\nu_2, \mu'_2q_2\nu'_2)$ alors

$$(\mu_1q_1\nu_1, \mu'_1q_1\nu'_1) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (\mu_2q_2\nu_2, \mu'_2q_2\nu'_2)\ .$$

Comme de plus $(\mu_2q_2\nu_2, \mu'_2q_2\nu'_2) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (w'q'_0, w'q'_0)$, on a finalement $(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w'q'_0, w'q'_0)$.
- Ou bien $w = u$, $q_1 \in Q_u$ et $(\mu_1q_1\nu_1, \mu'_1q_1\nu'_1) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (uq'_0, uq'_0)$. Ce cas se traite comme le précédent en considérant le point 3 du lemme 5.4.
- Ou bien $(\mu_1q_1\nu_1, \mu'_1q_1\nu'_1)$ est tel que $\mu'_1 = w$, $\nu'_1 = \varepsilon$ et $(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu_1q_1\nu_1, \mu'_1q_1\nu'_1)$. On a alors en résumé dans $\mathcal{G}(\mathcal{M})$:

$$(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu_1q_1\nu_1, \mu'_1q_1\nu'_1)$$

et

$$(\mu_1q_1\nu_1, \mu'_1q_1\nu'_1) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (\mu_2q_2\nu_2, \mu'_2q_2\nu'_2)$$

et

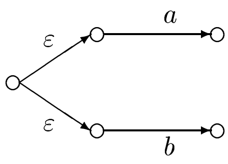
$$(\mu_2q_2\nu_2, \mu'_2q_2\nu'_2) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (w'q'_0, w'q'_0)$$

c'est-à-dire $(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w'q'_0, w'q'_0)$.

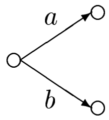
Finalement, dans chacun des cas mentionnés ci-dessus, on a bien $(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w'q'_0, w'q'_0)$ et donc, d'après le lemme 5.5, $w' \in (wa){\downarrow_{\mathcal{S}}}$ *i.e.* $w \xrightarrow[\mathcal{G}(\mathcal{T})]{a} w'$ avec $w' \leftrightsquigarrow (\mu_2q_2\nu_2, \mu'_2q_2\nu'_2)$.

Le lecteur attentif aura noté que dans cette démonstration nous ne considérons que des arcs de $\mathcal{G}(\mathcal{T})$. Nous ne perdons aucune généralité car comme dans ce graphe il n'y a pas d'ε-transitions, pour tous $w, w' \in R$ on a $w \underset{\mathcal{G}(\mathcal{T})}{\overset{a}{\dashrightarrow}} w'$ si et seulement si $w \underset{\mathcal{G}(\mathcal{T})}{\overset{a}{\longrightarrow}} w'$. $\square$

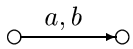
Notons qu'il existe des graphes qui ne sont ε-équivalents à aucun graphe sans ε-transition. Considérons par exemple le graphe $\mathcal{G}$ suivant :



D'après la définition de l'ε-équivalence, un graphe sans ε-transition qui serait ε-équivalent à $\mathcal{G}$ aurait forcément l'une des deux formes suivantes :



ou



Clairement, aucun de ces deux graphes n'est ε-équivalent à $\mathcal{G}$. Par conséquent, comme il n'y a pas d'ε-transition dans le graphe d'une spécification de Thue, il n'existe aucune spécification de Thue dont le graphe serait ε-équivalent à $\mathcal{G}$.

Enfin, dans le cas où la spécification de Thue $\mathcal{T}$ dont on est parti au début de ce paragraphe est *linéaire*, *i.e.* est telle qu'il existe une fonction linéaire $f : \mathbb{N} \to \mathbb{N}$ vérifiant :

$$\forall w, w' \in \Sigma^*,\ w \underset{\mathcal{S}}{\overset{*}{\rightarrow}} w' \Rightarrow |w'| \leq f\big(|w|\big) \ ,$$

alors la machine $\mathcal{M}$ construite ci-dessus est linéairement bornée (car le fonctionnement de $\mathcal{M}$ consiste à appliquer les règles de réécritures de $\mathcal{S}$).

Le théorème suivant résume tous ces résultats (nous rappelons que $\mathcal{G}[\text{TS}]$ désigne la classe des graphes des spécifications de Thue et $\mathcal{G}[\text{Lin-TS}]$ celle des graphes des spécifications de Thue linéaires).

Théorème 5.7 *Les classes des graphes des machines de Turing et des spécifications de Thue sont liées de la façon suivante.*

- $\mathcal{G}[\text{TS}]$ *est ε-équivalente à un sous-ensemble strict de* $\mathcal{G}[\text{TM}]$.
- $\mathcal{G}[\text{Lin-TS}]$ *est ε-équivalente à un sous-ensemble strict de* $\mathcal{G}[\text{Lin-TM}]$.

4.2 Un premier exemple

On considère la spécification de Thue

$$\mathcal{T} = \Big(\{a\},\ \{a, b\},\ \{ab \to \varepsilon, ab \to b\},\ a^*,\ \varepsilon\Big)$$

et l'automate fini, déterministe, complet et sans ε-transitions suivant qui reconnaît a^* :

$$\mathcal{A} = \Big(\{p_0\},\ \{a\},\ \{(p_0, a, p_0)\},\ p_0,\ \{p_0\}\Big)\ .$$

La machine de Turing associée à $\mathcal{T}$ est

$$\mathcal{M} = \Big(Q_{\mathcal{M}},\ \{a\},\ \{a, b, \square\},\ \{a, \square\},\ \delta_{\mathcal{M}},\ q_0\Big)$$

où $Q_{\mathcal{M}}$ est l'ensemble des états qui interviennent dans $\delta_{\mathcal{M}}$ et

$$\delta_{\mathcal{M}} = \delta_u \cup \delta_{\text{départ}} \cup \delta_{\text{vérifier-}wa} \cup \delta_{\text{réécrire}} \cup \delta_{\text{décaler}} \cup \delta_{\text{vérifier}} \cup \delta_{\text{remplacer}} \cup \delta_{\text{chercher}} \cup \delta_{\text{retour}}\ .$$

Les ensembles de transitions qui composent $\delta_{\mathcal{M}}$ sont les suivants.

– δ_u permet d'inscrire u, *i.e.* ε, sur les bandes B_1 et B_2 et de passer en configuration (uq_0', uq_0') :

$$\delta_u = \Big\{\big(q_0, \varepsilon, \square, \square, \square, \blacksquare, \square, \blacksquare, q_0'\big)\Big\}.$$

– $\delta_{\text{départ}}$ est l'ensemble :

$$\Big\{\big(q_0', \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, [\overline{a}, \in]\ \big),\ \big(q_0', \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, [b, q_0']\ \big)\Big\}.$$

– $\delta_{\text{vérifier-}wa}$ est l'ensemble :

$$\Big\{\quad \big(\ [\overline{a}, \in], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [\overline{a}, \in]\ \big), \big(\ [\overline{a}, \in], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacksquare, [\overline{a}, p_0]\ \big),$$
$$\big(\ [\overline{a}, p_0], \varepsilon, a, \square, a, \blacktriangleright, \square, \blacksquare, [\overline{a}, p_0]\ \big), \big(\ [\overline{a}, p_0], a, \square, \square, a, \blacktriangleright, a, \blacktriangleright, q_0'\big) \quad \Big\}.$$

– $\delta_{\text{réécrire}}$ est l'ensemble :

$$\Big\{\quad \big(\ [b, q_0'], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [b, ab \to \varepsilon \mid ab]\ \big), \big(\ [b, q_0'], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [b, ab \to b \mid ab]\ \big),$$
$$\big(\ [b, ab \to \varepsilon \mid b], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [b, ab \to \varepsilon \mid ab]\ \big),$$
$$\big(\ [b, ab \to b \mid b], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [b, ab \to b \mid ab]\ \big),$$
$$\big(\ [b, ab \to \varepsilon \mid ab], \varepsilon, a, \square, a, \blacktriangleright, \square, \blacksquare, [b \mid ab, \varepsilon]\ \big), \big(\ [b, ab \to b \mid ab], \varepsilon, a, \square, a, \blacktriangleright, \square, \blacksquare, [b \mid ab, b]\ \big),$$
$$\big(\ [b, ab \to \varepsilon \mid ab], \varepsilon, b, \square, b, \blacktriangleright, \square, \blacksquare, [b \mid ab, \varepsilon]\ \big), \big(\ [b, ab \to b \mid ab], \varepsilon, b, \square, b, \blacktriangleright, \square, \blacksquare, [b \mid ab, b]\ \big),$$
$$\big(\ [b, ab \to \varepsilon \mid ab], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacksquare, [b \mid ab, \varepsilon]\ \big), \big(\ [b, ab \to b \mid ab], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacksquare, [b \mid ab, b]\ \big),$$
$$\big(\ [b \mid ab, \varepsilon], \varepsilon, a, \square, a, \blacksquare, \square, \blacksquare, [b, \blacktriangleleft\ \mid ab]\ \big), \big(\ [b \mid ab, b], \varepsilon, a, \square, b, \blacktriangleright, \square, \blacksquare, [b \mid b, \varepsilon]\ \big),$$
$$\big(\ [b \mid b, \varepsilon], \varepsilon, b, \square, b, \blacksquare, \square, \blacksquare, [b, \blacktriangleleft\ \mid b]\ \big) \quad \Big\}.$$

Le lecteur aura noté que selon la définition de cet ensemble, il faudrait également considérer des transitions identiques où des états de la forme $[a \ldots]$ viendraient remplacer les états de la forme $[b\ldots]$. Ces transitions n'ont en fait aucune utilité car la lettre a est telle que $R.a \subseteq R$ *i.e.* pour tout mot w de R, wa est irréductible par $\mathcal{S}$. Or, les transitions de cet ensemble dont l'origine et le but sont des états de la forme $[a\ldots]$ doivent servir à réécrire wa.

– Lorsque $\mathcal{M}$ applique la règle $ab \to \varepsilon$ à une inscription de B_1 ayant la forme w_1abw_2, elle transforme ce mot en $w_1\square\square w_2$ puis décale w_2 vers la gauche pour que l'inscription de B_1 deviennent w_1w_2. Ces opérations débutent à l'état $[b, \blacktriangleleft \mid ab]$ et se terminent à l'état $[b, \in]$.

  De même, lorsque $\mathcal{M}$ applique la règle $ab \to b$ à une inscription de B_1 ayant la forme w_1abw_2, elle transforme ce mot en w_1bbw_2 (le a est remplacé par le membre gauche de la règle), puis en $w_1b\square w_2$. Ensuite, elle décale vers la gauche le mot w_2 pour que l'inscription de B_1 deviennent w_1bw_2. Les opérations de transformation $w_1bbw_2 \to w_1b\square w_2$ et de décalage débutent à l'état $[b, \blacktriangleleft \mid b]$ et se terminent à l'état $[b, \in]$.

  Ce sont les transitions de $\delta_{\text{décaler}}$ qui permettent d'effectuer ces opérations. Nous ne donnons pas le détail de cet ensemble.

– $\delta_{\text{vérifier}}$ est l'ensemble :

$$\Big\{ \ \big(\,[b,\in], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [b,\in]\,\big), \big(\,[b,\in], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacksquare, [b,p_0]\,\big),$$
$$\big(\,[b,p_0], \varepsilon, a, \square, a, \blacktriangleright, \square, \blacksquare, [b,p_0]\,\big), \big(\,[b,p_0], b, \square, \square, \square, \blacktriangleleft, \square, \blacktriangleleft, [\text{remplacer}]\,\big),$$
$$\big(\,[b,p_0], \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, [b]\,\big) \qquad \Big\}.$$

  Vu la remarque précédente concernant les transitions de $\delta_{\text{réécrire}}$ faisant intervenir des états de la forme $[a\ldots]$, il est suffisant de ne considérer que les transitions de $\delta_{\text{vérifier}}$ dont l'origine et le but sont des états de la forme $[b\ldots]$.

– $\delta_{\text{remplacer}}$ permet d'inscrire le mot de B_1 sur B_2 et est l'ensemble :

$$\Big\{ \ \big(\,[\text{remplacer}], \varepsilon, a, a, a, \blacktriangleleft, a, \blacktriangleleft, [\text{remplacer}]\,\big),$$
$$\big(\,[\text{remplacer}], \varepsilon, a, \square, a, \blacktriangleleft, a, \blacktriangleleft, [\text{remplacer}]\,\big),$$
$$\big(\,[\text{remplacer}], \varepsilon, \square, a, \square, \blacksquare, a, \blacksquare, [\text{effacer}]\,\big),\ \big(\,[\text{remplacer}], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacktriangleright, [\text{aller fin}]\,\big),$$
$$\big(\,[\text{effacer}], \varepsilon, \square, a, \square, \blacksquare, \square, \blacktriangleleft, [\text{effacer}]\,\big),\ \big(\,[\text{effacer}], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacktriangleright, [\text{aller fin}]\,\big),$$
$$\big(\,[\text{aller fin}], \varepsilon, a, a, a, \blacktriangleright, a, \blacktriangleright, [\text{aller fin}]\,\big),\ \big(\,[\text{aller fin}], \varepsilon, \square, \square, \square, \blacksquare, \square, \blacksquare, q'_0\big) \qquad \Big\}.$$

– δ_{chercher} est l'ensemble :

$$\Big\{ \ \big(\,[b], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [b]\,\big),\ \big(\,[b], \varepsilon, a, \square, a, \blacktriangleright, \square, \blacksquare, [b]\,\big),$$
$$\big(\,[b], \varepsilon, b, \square, b, \blacktriangleleft, \square, \blacksquare, [b]\,\big),\ \big(\,[b], \varepsilon, b, \square, b, \blacktriangleright, \square, \blacksquare, [b]\,\big),$$
$$\big(\,[b], \varepsilon, b, \square, b, \blacktriangleleft, \square, \blacksquare, [b, ab \to \varepsilon \mid b]\,\big),\ \big(\,[b], \varepsilon, b, \square, b, \blacktriangleleft, \square, \blacksquare, [b, ab \to b \mid b]\,\big) \ \Big\}.$$

Il est suffisant de ne considérer que les transitions de δ_{chercher} faisant intervenir des états de la forme $[b\ldots]$.

– δ_{retour} permet à la machine de revenir à la configuration qu'elle avait avant de commencer à réécrire un mot. Cet ensemble est composé de transitions qui débutent à partir de tous les états qui interviennent dans $\delta_{\text{vérifier-}wa}$, $\delta_{\text{réécrire}}$, $\delta_{\text{décaler}}$, $\delta_{\text{vérifier}}$ et δ_{chercher}.

4.3 Spécification de Thue associée à une machine de Turing

Comme nous l'avons montré ci-dessus, il n'est pas toujours possible d'associer une spécification de Thue à une machine de Turing de telle sorte que leurs graphes soient ε-équivalents. L'idée ici est d'obtenir un résultat plus précis que celui du théorème 5.7, *i.e.* de caractériser plus exactement la classe $\mathcal{G}[\text{TS}]$ à l'intérieur de $\mathcal{G}[\text{TM}]$. Nous construisons ci-dessous une spécification de Thue à partir d'une machine de Turing. Lorsque cette machine vérifie trois conditions particulières que nous précisons ci-après, son graphe est ε-équivalent à celui de la spécification. Pour énoncer ces conditions, nous avons besoin des définitions suivantes : étant donné un graphe $\mathcal{G}$ étiqueté par Σ,

– on appelle *ε-chemin* tout chemin de $\mathcal{G}$ étiqueté par ε et

– pour tout sommet v de $\mathcal{G}$ et toute lettre a de Σ, l'ensemble $s_a(v)$ est défini par :

$$s_a(v) = \left\{v' \mid \exists v'',\ v \underset{\mathcal{G}}{\overset{\varepsilon}{\dashrightarrow}} v'' \underset{\mathcal{G}}{\overset{a}{\longrightarrow}} v'\right\} .$$

Soit $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0)$ une machine de Turing à une bande de travail dont les ensembles Q, Σ et Γ sont deux à deux disjoints et dont le graphe vérifie les conditions suivantes :

1. aucun arc ayant pour origine la racine distinguée q_0 de $\mathcal{G}(\mathcal{M})$ n'est étiqueté par ε,
2. $\mathcal{G}(\mathcal{M})$ ne contient aucun ε-chemin infini et
3. pour tout ε-chemin P dans $\mathcal{G}(\mathcal{M})$, pour tous sommets v et v' dans P et tout a dans Σ on a $s_a(v) = s_a(v')$.

L'ensemble des configurations internes de $\mathcal{M}$ est inclus dans $\Gamma^*\ Q\ \Gamma^*$ et sa configuration initiale est q_0. Soit \$ et £ deux nouveaux symboles n'appartenant pas à $Q \cup \Sigma \cup \Gamma$. On considère la spécification de Thue $\mathcal{T} = \big(\Gamma \cup Q \cup \{\$, \pounds\}, \Sigma, \mathcal{S}, R, u\big)$ telle que :

– $\mathcal{S}$ est l'ensemble $\mathcal{S}_Q \ \cup\ \mathcal{S}_{\blacktriangleleft} \ \cup\ \mathcal{S}_{\blacktriangleright} \ \cup\ \mathcal{S}_{\blacksquare}$ où

$$\mathcal{S}_Q = \big\{\pounds a \to \pounds \mid a \in \Sigma\big\} \ \cup\ \big\{Xa \to aX \mid a \in \Sigma,\ X \in \Gamma\big\} \ \cup\ \big\{\$\square \to \$, \square\pounds \to \pounds\big\},$$

$\mathcal{S}_{\blacktriangleleft}$ est l'ensemble des règles qui traitent le cas des transitions de $\mathcal{M}$ qui déplacent la tête de la bande de travail vers la gauche :

$$\begin{aligned}
\mathcal{S}_{\blacktriangleleft} \quad = \quad & \big\{ ZqaX \to q'ZY \mid a \in \Sigma,\ Z \in \Gamma,\ (q, a, X, Y, \blacktriangleleft, q') \in \delta \big\} \cup \\
& \big\{ \$qa£ \to \$q'Y£ \mid a \in \Sigma,\ (q, a, \square, Y, \blacktriangleleft, q') \in \delta \big\} \cup \\
& \big\{ Zqa£ \to q'ZY£ \mid a \in \Sigma,\ Z \in \Gamma,\ (q, a, \square, Y, \blacktriangleleft, q') \in \delta \big\} \cup \\
& \big\{ \$qaX \to \$q'\square Y \mid a \in \Sigma,\ (q, a, X, Y, \blacktriangleleft, q') \in \delta \big\} \cup \\
& \big\{ ZqX \to q'ZY \mid Z \in \Gamma,\ (q, \varepsilon, X, Y, \blacktriangleleft, q') \in \delta \big\} \cup \\
& \big\{ \$q£ \to \$q'Y£ \mid (q, \varepsilon, \square, Y, \blacktriangleleft, q') \in \delta \big\} \cup \\
& \big\{ Zq£ \to q'ZY£ \mid Z \in \Gamma,\ (q, \varepsilon, \square, Y, \blacktriangleleft, q') \in \delta \big\} \cup \\
& \big\{ \$qX \to \$q'\square Y \mid (q, \varepsilon, X, Y, \blacktriangleleft, q') \in \delta \big\},
\end{aligned}$$

$\mathcal{S}_{\blacktriangleright}$ est l'ensemble des règles qui traitent le cas des transitions de $\mathcal{M}$ qui déplacent la tête de la bande de travail vers la droite :

$$\begin{aligned}
\mathcal{S}_{\blacktriangleright} \quad = \quad & \big\{ qaX \to Yq' \mid a \in \Sigma,\ (q, a, X, Y, \blacktriangleright, q') \in \delta \big\} \cup \\
& \big\{ qa£ \to Yq'£ \mid a \in \Sigma,\ (q, a, \square, Y, \blacktriangleright, q') \in \delta \big\} \cup \\
& \big\{ qX \to Yq' \mid (q, \varepsilon, X, Y, \blacktriangleright, q') \in \delta \big\} \cup \\
& \big\{ q£ \to Yq'£ \mid (q, \varepsilon, \square, Y, \blacktriangleright, q') \in \delta \big\},
\end{aligned}$$

et $\mathcal{S}_{\blacksquare}$ est l'ensemble des règles qui traitent le cas des transitions de $\mathcal{M}$ qui ne déplacent pas la tête de la bande de travail :

$$\begin{aligned}
\mathcal{S}_{\blacksquare} \quad = \quad & \big\{ qaX \to q'Y \mid a \in \Sigma,\ (q, a, X, Y, \blacksquare, q') \in \delta \big\} \cup \\
& \big\{ qa£ \to q'Y£ \mid a \in \Sigma,\ (q, a, \square, Y, \blacksquare, q') \in \delta \big\} \cup \\
& \big\{ qX \to q'Y \mid (q, \varepsilon, X, Y, \blacksquare, q') \in \delta \big\} \cup \\
& \big\{ q£ \to q'Y£ \mid (q, \varepsilon, \square, Y, \blacksquare, q') \in \delta \big\} \ ,
\end{aligned}$$

- $R = \big(\$\ \Gamma^*\ Q\ \Gamma^*\ £\big) \cap \mathrm{Irr}(\mathcal{S})$ (notons que R est rationnel car $\$\ \Gamma^*\ Q\ \Gamma^* £$ et $\mathrm{Irr}(\mathcal{S})$ le sont) et
- $u = \$q_0£$ (la condition 1 imposée au graphe de $\mathcal{M}$ et le lemme 5.8 assurent que $u \in R$).

Le résultat suivant est une conséquence directe de la définition de $\mathcal{T}$.

Lemme 5.8 *Les deux faits suivants sont vérifiés.*

1. $R = \Big\{\ \$\mu q \nu £ \bigm| \mu q \nu \in \Gamma^*\ Q\ \Gamma^* \quad et \quad \delta \cap \big(\{q\} \times \{\varepsilon\} \times \{X\} \times \Gamma \times \{\blacktriangleleft, \blacktriangleright, \blacksquare\} \times Q\big) = \varnothing \Big\}$
   où le symbole X représente la première lettre de ν si ce mot n'est pas vide et $\square$ sinon.

2. *Soit ξ et ξ' des configurations de $\mathcal{M}$ telles que $\$\xi\pounds \in R$ et $\$\xi'\pounds \in R$. Soit a une lettre de Σ. Alors, il existe une configuration ξ'' telle que $\xi \xrightarrow[\mathcal{G}(\mathcal{M})]{a} \xi'' \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \xi'$ si et seulement si $\$\xi\pounds \xrightarrow[\mathcal{G}(\mathcal{T})]{a} \$\xi'\pounds$.*

Proposition 5.9 *Les graphes $\mathcal{G}(\mathcal{M})$ et $\mathcal{G}(\mathcal{T})$ sont ε-équivalents.*

Preuve. On appelle $V_{\mathcal{T}}$ l'ensemble des sommets de $\mathcal{G}(\mathcal{T})$ et $V_{\mathcal{M}}$ celui de $\mathcal{G}(\mathcal{M})$. Soit $\leftrightsquigarrow \subseteq \Gamma^* \ Q \ \Gamma^* \times R$ la relation définie par :

$$\leftrightsquigarrow = \left\{(\mu q\nu, \$\mu' q'\nu'\pounds) \mid \mu q\nu \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu' q'\nu' \text{ et } \forall v \in V_{\mathcal{M}},\ \mu' q'\nu' \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v \Rightarrow a \neq \varepsilon\right\} .$$

Dans cette démonstration, le terme « ε-arc » signifie « arc étiqueté par ε ». Les points suivants sont vérifiés.

- $\mathcal{D}\mathrm{om}(\leftrightsquigarrow) = V_{\mathcal{M}}$ car on a supposé qu'il n'y a aucun ε-chemin infini dans $\mathcal{G}(\mathcal{M})$.
- $\mathcal{R}\mathrm{an}(\leftrightsquigarrow) = V_{\mathcal{T}}$. Soit $\mu q\nu$ et $\mu' q'\nu'$ des éléments de $\Gamma^* \ Q \ \Gamma^*$ tels que $\mu q\nu \leftrightsquigarrow \$\mu' q'\nu'\pounds$. Alors, on a $\mu q\nu \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu' q'\nu'$ et $\mu' q'\nu'$ n'est l'origine d'aucun ε-arc de $\mathcal{G}(\mathcal{M})$. Comme la racine distinguée de $\mathcal{G}(\mathcal{M})$ n'est elle-même l'origine d'aucun ε-arc, il existe un sommet v de $\mathcal{G}(\mathcal{M})$ origine d'aucun ε-arc et une lettre $a \in \Sigma$ tels que l'on a une situation de la forme $v \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v' \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu q\nu$. Par conséquent, on a $v \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v' \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu q\nu \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu' q'\nu'$ *i.e.* $v \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v' \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu' q'\nu'$ avec $\$v\pounds$ et $\$\mu' q'\nu'\pounds$ des éléments de R. D'après le point 2 du lemme 5.8, on a donc $\$v\pounds \xrightarrow[\mathcal{G}(\mathcal{T})]{a} \$\mu' q'\nu'\pounds$ ce qui prouve que $\$\mu' q'\nu'\pounds$ est un sommet de $\mathcal{G}(\mathcal{T})$.
  Inversement, tout sommet $\$v'\pounds$ de $\mathcal{G}(\mathcal{T})$ est tel que soit il est la racine distinguée de $\mathcal{G}(\mathcal{T})$, auquel cas il est dans $\mathcal{R}\mathrm{an}(\leftrightsquigarrow)$, soit il existe $\$v\pounds$ dans $V_{\mathcal{T}}$ tel que $\$v\pounds \xrightarrow[\mathcal{G}(\mathcal{T})]{a} \$v'\pounds$. Dans ce cas, d'après le point 2 du lemme 5.8, il existe v'' tel que $v \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v'' \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} v'$. De plus, comme $\$v'\pounds \in R$, v' n'est l'origine d'aucun ε-arc de $\mathcal{G}(\mathcal{M})$. Finalement, on a $v'' \leftrightsquigarrow \$v'\pounds$ ce qui prouve que $\$v'\pounds$ est un élément de $\mathcal{R}\mathrm{an}(\leftrightsquigarrow)$.
- q_0 est la racine distinguée de $\mathcal{G}(\mathcal{M})$, $\$q_0\pounds$ est celle de $\mathcal{G}(\mathcal{T})$ et on a $q_0 \leftrightsquigarrow \$q_0\pounds$ car $q_0 \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} q_0$ et q_0 n'est l'origine d'aucun ε-arc.
- Soit $a \in \Sigma$, $\mu_1 q_1 \nu_1$ et $\mu_1' q_1' \nu_1'$ des sommets de $\mathcal{G}(\mathcal{M})$ tels que $\mu_1 q_1 \nu_1 \overset{a}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu_1' q_1' \nu_1'$ et $\$\mu_2 q_2 \nu_2\pounds$ un sommet de $\mathcal{G}(\mathcal{T})$ tel que $\mu_1 q_1 \nu_1 \leftrightsquigarrow \$\mu_2 q_2 \nu_2\pounds$.
  Le fait « $\mu_1 q_1 \nu_1 \overset{a}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu_1' q_1' \nu_1'$ » est équivalent à l'existence de deux sommets v et v' de $\mathcal{G}(\mathcal{M})$ tels que $\mu_1 q_1 \nu_1 \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} v \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v' \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu_1' q_1' \nu_1'$. De plus, comme $\mu_1 q_1 \nu_1 \leftrightsquigarrow \$\mu_2 q_2 \nu_2\pounds$, $\mu_1 q_1 \nu_1$ et $\mu_2 q_2 \nu_2$ se trouvent sur un même ε-chemin, ce qui implique, comme $s_a(\mu_1 q_1 \nu_1) = s_a(\mu_2 q_2 \nu_2)$ (condition3 imposée à $\mathcal{M}$), qu'il existe v'' tel que $\mu_2 q_2 \nu_2 \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} v'' \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v'$ (par définition de l'application s_a). Comme $\mu_2 q_2 \nu_2$ n'est l'origine d'aucun ε-arc (car $\mu_1 q_1 \nu_1 \leftrightsquigarrow \$\mu_2 q_2 \nu_2\pounds$), on a plus précisément $v'' = \mu_2 q_2 \nu_2$ et $\mu_2 q_2 \nu_2 \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v'$.
  Soit $\$\xi\pounds$ un sommet de $\mathcal{G}(\mathcal{T})$ tel que $\mu_1' q_1' \nu_1' \leftrightsquigarrow \$\xi\pounds$ (ce sommet existe car $\mathcal{D}\mathrm{om}(\leftrightsquigarrow) = V_{\mathcal{M}}$ et $\mathcal{R}\mathrm{an}(\leftrightsquigarrow) = V_{\mathcal{T}}$). On a alors pour résumer :

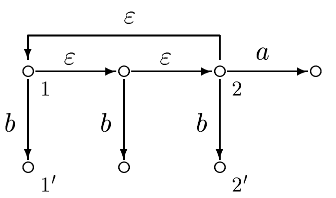
$$\mu_2 q_2 \nu_2 \xrightarrow[\mathcal{G}(\mathcal{M})]{a} v' \quad \text{et} \quad v' \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \mu_1' q_1' \nu_1' \quad \text{et} \quad \mu_1' q_1' \nu_1' \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} \xi ,$$

c'est-à-dire $\mu_2 q_2 \nu_2 \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} \xi$ où $\$\mu_2 q_2 \nu_2 \pounds$ et $\$\xi\pounds$ sont des mots de R. Par conséquent, d'après le point 2 du lemme 5.8, on a $\$\mu_2 q_2 \nu_2 \pounds \underset{\mathcal{G}(\mathcal{T})}{\overset{a}{\longrightarrow}} \$\xi\pounds$ avec, rappelons-le, $\mu_1 q_1 \nu_1 \leftrightsquigarrow \$\mu_2 q_2 \nu_2 \pounds$ et $\mu_1' q_1' \nu_1' \leftrightsquigarrow \$\xi\pounds$.

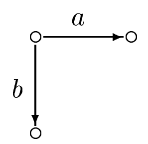
– Soit $a \in \Sigma$, $\$\mu_2 q_2 \nu_2 \pounds$ et $\$\mu_2' q_2' \nu_2' \pounds$ des sommets de $\mathcal{G}(\mathcal{T})$ tels que $\$\mu_2 q_2 \nu_2 \pounds \underset{\mathcal{G}(\mathcal{T})}{\overset{a}{\longrightarrow}} \$\mu_2' q_2' \nu_2' \pounds$ et $\mu_1 q_1 \nu_1$ un sommet de $\mathcal{G}(\mathcal{M})$ tel que $\mu_1 q_1 \nu_1 \leftrightsquigarrow \$\mu_2 q_2 \nu_2 \pounds$. Alors, par définition de $\leftrightsquigarrow$, $\mu_1 q_1 \nu_1 \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} \mu_2 q_2 \nu_2$ et, d'après le lemme 5.8, $\mu_2 q_2 \nu_2 \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} \mu_2' q_2' \nu_2'$. Par conséquent, on a $\mu_1 q_1 \nu_1 \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} \mu_2' q_2' \nu_2'$ avec $\mu_1 q_1 \nu_1 \leftrightsquigarrow \$\mu_2 q_2 \nu_2 \pounds$ et $\mu_2' q_2' \nu_2' \leftrightsquigarrow \$\mu_2' q_2' \nu_2' \pounds$. □

Notons que nous ne perdons aucune généralité en considérant des machines de Turing à une seule bande de travail. En effet, comme cela est expliqué au paragraphe 2 de ce chapitre, il est toujours possible d'associer à une machine à k bandes de travail une machine à une seule bande telle que les graphes associés aux deux machines sont ε-équivalents.

Enfin, les trois conditions énoncées au début de cette partie ne sont pas nécessaires. Il suffit de considérer les graphes suivants :



et



Ils sont ε-équivalents mais celui de gauche ne vérifie aucune des trois conditions : sa racine distinguée 1 est l'origine d'un ε-arc, il contient des ε-chemins infinis et les sommets 1 et 2 appartiennent à un même ε-chemin et sont tels que $s_b(1) \neq s_b(2)$ (car $1' \in s_b(1)$, $1' \notin s_b(2)$ et $1'$ et $2'$ sont des sommets distincts). De plus, ce graphe est celui d'une machine de Turing et l'autre celui d'une spécification de Thue.

Le premier de ces deux graphes vérifie la condition fournie par la conjecture suivante et qui est plus générale que la condition 3 énoncée au début de cette partie.

Conjecture 5.10 *Soit $\mathcal{G}$ un graphe étiqueté par Σ. Pour tout sommet v de $\mathcal{G}$ et tout mot w dans Σ^*, on définit l'ensemble $s_w(v)$ par $s_w(v) = \{v' \mid v \underset{\mathcal{G}}{\overset{w}{\dashrightarrow}} v'\}$.*

Supposons que pour tout ε-chemin P dans $\mathcal{G}$, tous sommets v et v' dans P et tout w dans Σ^* on ait $s_w(v) \neq \varnothing \Rightarrow s_w(v') \neq \varnothing$. Alors, $\mathcal{G}$ est ε-équivalent à un graphe sans ε-transition.

Nous terminons cette partie par la remarque suivante. Dans le cas de machines de Turing temps-réel, les conditions énoncées au début de cette partie sont toujours vérifiées et la construction détaillée ci-dessus conduit à des spécifications de Thue longueur-décroissantes[6]. Nous rappelons au lecteur que $\mathcal{G}$[Real-TM] désigne la classe des graphes des machines de Turing temps-réel et $\mathcal{G}$[Len-TS] celle des graphes des spécifications de Thue longueur-décroissantes. Nous pouvons donc énoncer le résultat suivant.

Théorème 5.11 $\mathcal{G}$[Real-TM] *est ε-équivalent à un sous-ensemble de* $\mathcal{G}$[Len-TS].

6. Nous avons vu au chapitre 4 qu'une spécification de Thue est longueur-décroissante si son semi-système de Thue $\mathcal{S} \subseteq \Sigma^* \times \Sigma^*$ vérifie : $\forall w, w' \in \Sigma^*$, $w \underset{\mathcal{S}}{\overset{*}{\rightarrow}} w' \Rightarrow |w'| \leq |w|$.

4.4 Un deuxième exemple

On considère la machine de Turing

$$\Big(\{q_1, q_2, q_\varepsilon\},\ \{a\},\ \{a, b, \square\},\ \delta,\ q_1\ \Big)$$

où δ est l'ensemble

$$\Big\{(q_1, a, \square, a, \blacktriangleright, q_2),\ (q_2, a, \square, a, \blacktriangleright, q_\varepsilon),\ (q_\varepsilon, \varepsilon, \square, b, \blacktriangleright, q_1)\Big\}\ .$$

Pour toute entrée de la forme a^{2p}, cette machine inscrit $(aab)^p$ sur sa bande de travail et pour toute entrée de la forme a^{2p+1}, elle inscrit $(baa)^p a$ sur sa bande de travail (c'est-à-dire que dans tous les cas de figure, elle recopie son entrée sur sa bande de travail en insérant un b tous les deux a). Le graphe associé à $\mathcal{M}$ est

$$\boxed{q_1} \xrightarrow{\quad a \quad} \boxed{aq_2} \xrightarrow{\quad a \quad} \boxed{aaq_\varepsilon} \xrightarrow{\quad \varepsilon \quad} \boxed{aabq_1} \quad ------$$

et vérifie les trois conditions énoncées au début du paragraphe 4.3 : sa racine distinguée q_1 n'est l'origine d'aucun ε-arc, il ne contient aucune ε-chemin infini et pour tous sommets v et v' d'un même ε-chemin, on a $s_a(v) = s_a(v')$. La spécification de Thue associée à $\mathcal{M}$ est

$$\mathcal{T} = \Big(\{q_\varepsilon, q_1, q_2\} \cup \{a, b, \square\} \cup \{\$, \pounds\},\ \{a\},\ \mathcal{S},\ \$\,\mathrm{I}\,\pounds,\ \$\ q_1\ \pounds\Big)$$

où $\mathrm{I} = \{a, b, \square\}^*.\{q_\varepsilon, q_1, q_2\}.\{a, b, \square\}^*$ et $\mathcal{S}$ est l'ensemble suivant :

$$\Big\{\pounds a \to a\pounds,\ \square a \to a\square,\ ba \to ab, \$\square \to \$,\ \square\pounds \to \pounds\Big\} \bigcup \mathcal{S}_1 \bigcup \mathcal{S}_2 \bigcup \mathcal{S}_\varepsilon$$

où

$$\mathcal{S}_1 = \Big\{q_1 a\square \to aq_2,\ q_1 a\pounds \to aq_2\pounds\Big\}$$

est l'ensemble des règles engendré par la transition $(q_1, a, \square, a, \blacktriangleright, q_2)$,

$$\mathcal{S}_2 = \Big\{q_2 a\square \to aq_\varepsilon,\ q_2 a\pounds \to aq_\varepsilon\pounds\Big\}$$

est l'ensemble des règles engendré par la transition $(q_2, a, \square, a, \blacktriangleright, q_\varepsilon)$ et

$$\mathcal{S}_\varepsilon = \Big\{q_\varepsilon\square \to bq_1,\ q_\varepsilon\pounds \to bq_1\pounds\Big\}$$

est l'ensemble des règles engendré par la transition $(q_\varepsilon, \varepsilon, \square, b, \blacktriangleright, q_1)$.

5 Théorie du premier ordre des graphes des machines de Turing

Jusqu'ici dans ce chapitre, nous nous sommes plutôt intéressés à la spécification de processus communiquants grâce à une approche basée sur les machines de Turing. Nous abordons maintenant le problème de la vérification formelle au sein de cette approche. De façon plus précise, nous supposons qu'un système de processus communiquants a été spécifié, c'est-à-dire que chaque processus séquentiel a été décrit par une machine de Turing et que les interactions ont été exprimées sous la forme d'une contrainte de synchronisation. Un tel système peut être représenté, à ε-équivalence près, par une machine de Turing dont le graphe est le produit synchronisé de ceux des processus qui composent le système. Le problème de la vérification consiste alors à calculer la valeur de vérité d'une formule d'une certaine logique sur le produit des graphes.

Nous affirmons que même pour des logiques assez faibles, il ne faut pas s'attendre à des solutions algorithmiques à ce problème de vérification mais plutôt à des solutions semi-algorithmiques. Plus précisément, nous montrons que la théorie du premier ordre des graphes des machines de Turing linéairement bornées n'est ni décidable ni même semi-décidable.

Comme cela est expliqué dans [KP99], afin de prouver ce résultat, nous devons introduire la notion d'automate linéairement borné. Formellement, un automate linéairement borné $\mathcal{L}_f$ est une machine linéairement bornée à une seule bande de travail[7] $\mathcal{L} = (Q, \Sigma, \Gamma, \delta, q_0)$ pourvue d'un état particulier $f \in Q$ appelé *état acceptant*. Le langage reconnu par $\mathcal{L}_f$ est noté $\mathrm{Lang}(\mathcal{L}_f)$ et est l'ensemble des étiquettes de tous les chemins de $\mathcal{G}(\mathcal{L}_f)$ qui vont de la configuration initiale jusqu'à une configuration de la forme $\mu f \nu$. Deux automates linéairement bornés $\mathcal{L}_f$ et $\mathcal{L}'_{f'}$ sont *équivalents* si $\mathrm{Lang}(\mathcal{L}_f) = \mathrm{Lang}(\mathcal{L}'_{f'})$.

Le problème $\mathscr{P}$:

Instance : Un automate linéairement borné $\mathcal{L}_f$.
Question : $\mathrm{Lang}(\mathcal{L}_f) = \varnothing$?

est bien connu pour ne pas être récursivement énumérable (voir par exemple dans [HU79] l'exercice 9.13 page 230, sa solution et le corollaire 1 page 192). Nous allons prouver le théorème suivant en utilisant ce fait. Nous rappelons auparavant au lecteur qu'un *énoncé* est une formule sans variable libre.

Théorème 5.12 *Le problème*

Instance : *Une machine linéairement bornée $\mathcal{L}$ et un énoncé φ de la logique du premier ordre associée à $|\mathcal{G}(\mathcal{L})|_1$.*
Question : *Est-ce que φ appartient à la théorie du premier ordre associée à $|\mathcal{G}(\mathcal{L})|_1$?*

n'est pas récursivement énumérable.

Preuve. Nous montrons que le problème $\mathscr{P}$ se réduit à celui énoncé dans le théorème.

Soit $\mathcal{L}_f = (Q, \Sigma, \Gamma, \delta, q_0, f)$ un automate linéairement borné. Soit $\mathcal{L}'_f = (Q', \Sigma, \Gamma, \delta', q_0, f)$ un automate équivalent à $\mathcal{L}_f$ et vérifiant les propriétés suivantes :

(1) $\delta' \cap (\{f\} \times \widetilde{\Sigma} \times \Gamma \times \Gamma \times \{◀, ▶, ■\} \times Q') = \varnothing$,

(2) $\delta' \cap (\{q\} \times \widetilde{\Sigma} \times \{X\} \times \Gamma \times \{◀, ▶, ■\} \times Q') \neq \varnothing$ pour tout $(q, X) \in (Q \smallsetminus \{f\}) \times \Gamma$.

Un tel automate linéairement borné peut être facilement construit à partir de $\mathcal{L}_f$. Concernant (1), pour chaque transition de δ qui a la forme $(q_1, c_1, X_1, Y_1, ♦_1, f)$, nous ajoutons à $\mathcal{L}_f$ une transition $(q_1, c_1, X_1, Y_1, ♦_1, q')$ où $q' \notin Q$ est un nouvel état. De plus, chaque transition $(f, c_2, X_2, Y_2, ♦_2, q_2) \in \delta$ est remplacée par $(q', c_2, X_2, Y_2, ♦_2, q_2)$. Concernant (2), nous ajoutons à $\mathcal{L}_f$ un nouvel état $p \notin Q$, les transitions $(p, \varepsilon, X, X, ■, p)$ pour tout $X \in \Gamma$ et pour chaque $(q, X) \in (Q \smallsetminus \{f\}) \times \Gamma$ tel que $\delta \cap (\{q\} \times \widetilde{\Sigma} \times \{X\} \times \Gamma \times \{◀, ▶, ■\} \times Q) = \varnothing$, la transition $(q, \varepsilon, X, X, ■, p)$.

7. Un automate linéairement borné peut avoir plusieurs bandes de travail, mais ici nous avons juste besoin des automates à une seule bande de travail.

Il est clair que $\mathrm{Lang}(\mathcal{L}_f) = \mathrm{Lang}(\mathcal{L}'_f)$ et qu'un sommet $\mu q \nu$ de $\mathcal{G}_{\mathcal{L}'_f}$ est bloquant si et seulement si $q = f$. Or, pour tout graphe $\mathcal{G}$ étiqueté par Σ, l'énoncé $\forall x \, \exists y \, \bigvee_{a \in \Sigma} \mathbf{arc}[a](x,y)$ appartient à la théorie du premier ordre associée à $|\mathcal{G}|_1$ si et seulement si $\mathcal{G}$ n'a pas de sommets bloquants. Par conséquent, $\mathrm{Lang}(\mathcal{L}_f) = \varnothing$ si et seulement si cet énoncé appartient à la théorie du premier ordre associée à $|\mathcal{G}(\mathcal{L}'_f)|_1$.
$\square$

Notons que le fait d'être un sommet bloquant est exprimable dans la logique de Hennessy–Milner. Par conséquent, le corollaire suivant se déduit directement de la preuve précédente.

Corollaire 5.13 *La logique de Hennessy–Milner n'est pas semi-décidable sur les graphes des machines linéairement bornées.*

Nous terminons ce paragraphe par la remarque suivante. Si l'existence d'un sommet bloquant est exprimable dans une logique, alors il ne peut exister de système formel complet pour vérifier la valeur de vérité de formules de cette logique sur les graphes des machines linéairement bornées.

Chapitre 6

Théorie du premier ordre du produit synchronisé

Sommaire

Nous avons introduit au chapitre 2, paragraphe 5, une définition particulière du produit synchronisé : dans le cas des graphes qui ont une racine distinguée, on se restreint à la partie accessible du produit synchronisé usuel. Dans ce chapitre, nous nous intéressons à la décidabilité de la théorie du premier ordre de cette variante et nous la comparons à celle du produit usuel. Les résultats obtenus dans les deux cas ne sont pas les mêmes.

1 Cas de graphes qui ont une racine distinguée

La décidabilité de la théorie du premier ordre n'est pas toujours conservée par le produit synchronisé que nous avons introduit. Il suffit de considérer les graphes de machines à pile qui, nous le rappelons, se situent au niveau le plus bas (pour l'inclusion) de la hiérarchie de graphes présentée au chapitre 3. Pour réaliser cette étude, nous avons besoin des définitions suivantes.

Un *automate à pile* $\mathcal{P}$ est une machine à pile $(Q, \Sigma, \Gamma, \delta, q_0, f)$ pourvue d'un état particulier $f \in Q$ appelé *état acceptant*. Une *configuration acceptante* de $\mathcal{P}$ est un élément de $\{f\}.\Gamma^*$.

Un mot $w \in \Sigma^*$ est *reconnu* par $\mathcal{P}$ lorsque w est l'étiquette d'un chemin dans $\mathcal{G}(\mathcal{P})$ depuis la configuration initiale q_0 jusqu'à une configuration acceptante. Le *langage* de $\mathcal{P}$ est l'ensemble de tous les mots reconnus par $\mathcal{P}$ et est noté $\mathrm{Lang}(\mathcal{P})$.

Nous montrons à présent que la théorie du premier ordre (sans quantification sur les arcs) des graphes des machines à pile n'est ni décidable ni même semi-décidable.

Théorème 6.1 *Le problème suivant n'est pas récursivement énumérable.*

Instance :	*n machines à pile $\mathcal{P}_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_{0i})$, $i \in [n]$, une contrainte de synchronisation $\mathscr{C}$ sur leur alphabet d'entrée et un énoncé φ de la logique du premier ordre associée à $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{P}_i)$.*
Question :	$\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{P}_i) \models \varphi$?

Preuve. Nous montrons que le problème

Instance :	n automates à pile temps-réel $\mathcal{P}_i$, $i \in [n]$, avec le même alphabet d'entrée.
Question :	$\bigcap_{i\in[n]} \mathrm{Lang}(\mathcal{P}_i) = \varnothing$?

qui est connu pour ne pas être récursivement énumérable (voir par exemple dans [HU79] le théorème 8.10, sa preuve et le corollaire 1 page 192), se réduit à celui énoncé dans le théorème.

Soit $\mathcal{P}_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q_{0i}, f_i)$, $i \in [n]$, n automates à pile temps-réel. Pour tout $i \in [n]$, soit $\mathcal{P}'_i = (Q'_i, \Sigma, \Gamma_i, \delta'_i, q_{0i}, f_i)$ un automate à pile vérifiant les propriétés suivantes.

1. $\delta'_i \cap (\{f_i\} \times \widetilde{\Sigma} \times \Gamma_i \times \Gamma_i^* \times Q_i) = \{(f_i, \varepsilon, X, X, f_i) \mid X \in \Gamma_i\}$,
2. $\delta'_i \cap (Q_i \setminus \{f_i\}) \times \{\varepsilon\} \times \Gamma_i \times \Gamma_i^* \times (Q_i \setminus \{f_i\}) = \varnothing$ et
3. $\mathrm{Lang}(\mathcal{P}'_i) = \mathrm{Lang}(\mathcal{P}_i)$.

Un tel automate peut être facilement construit à partir de $\mathcal{P}_i$. Concernant (1), pour chaque transition de δ_i qui a la forme (q, c, X, μ, f_i), nous ajoutons la transition (q, c, X, μ, q') où $q' \notin Q_i$ est un nouvel état, chaque transition $(f_i, c, X, \mu, q) \in \delta_i$ est remplacée par (q', c, X, μ, q) et nous ajoutons les transitions $(f_i, \varepsilon, X, X, f_i)$ pour tout $X \in \Gamma_i$. Il est clair que les points (2) et (3) sont alors également satisfaits.

À présent, considérons le produit $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{P}'_i)$ où $\mathscr{C}$ est la contrainte $\{(c, \ldots, c) \mid c \in \widetilde{\Sigma}\}$. Soit $\mathscr{L}$ l'ensemble des étiquettes des chemins de ce produit qui vont de la configuration $(q_{01}, \ldots, q_{0n})$ aux configurations de la forme $(\iota_1, \ldots, \iota_n)$ où, pour chaque $i \in [n]$, ι_i est une configuration acceptante de $\mathcal{P}'_i$. On a alors les résultats suivants.

1. $\iota \xrightarrow{(\varepsilon,\ldots,\varepsilon)} \iota'$ est un arc du produit si et seulement si $\iota = \iota'$ et chaque i^{e} coordonnée du n-uplet ι est une configuration acceptante de $\mathcal{P}'_i$ (ce résultat vient des points 1 et 2 vérifiés par les machines $\mathcal{P}'_i$) et
2. $\mathscr{L} = \big\{(w, \ldots, w) \mid w \in \bigcap_{i\in[n]} \mathrm{Lang}(\mathcal{P}_i)\big\}$ (ce résultat vient de la forme de la contrainte $\mathscr{C}$).

Par conséquent, si φ est la formule $\forall x\ \forall y\ \neg\mathbf{arc}[c](x, y)$, où c est le n-uplet $(\varepsilon, \ldots, \varepsilon)$, alors on a $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{G}(\mathcal{P}'_i) \models \varphi$ si et seulement si $\mathscr{L} = \varnothing$, *i.e.* si et seulement si $\bigcap_{i\in[n]} \mathrm{Lang}(\mathcal{P}_i) = \varnothing$. $\square$

2 Théorie du premier ordre du produit synchronisé usuel

Dans cette partie, nous ne distinguons plus de sommet particulier en tant que racine. Nous représentons donc les graphes au moyen de structures relationnelles sans le symbole de relation **r** et nous considérons le produit synchronisé usuel tel qu'introduit par Arnold et Nivat.

2.1 Quelques préliminaires

Étant donné un alphabet fini Σ, on considère l'ensemble $\mathcal{M}\mathrm{od}(\Sigma)$ des structures relationnelles sur l'ensemble de symboles de relations binaires $\{\mathbf{arc}[a] \mid a \in \widetilde{\Sigma}\}$.

Nous nous intéressons ici à la logique du premier ordre associée aux éléments de $\mathcal{M}\mathrm{od}(\Sigma)$. L'ensemble des formules de cette logique est noté $\mathcal{S}\mathrm{en}(\Sigma)$. Une *valuation* dans une structure $\mathcal{S}$ de $\mathcal{M}\mathrm{od}(\Sigma)$ est notée $\nu_{\mathcal{S}}$ et est une application de l'ensemble des variables dans le domaine de $\mathcal{S}$. La *relation de satisfaction* $\underset{\Sigma}{\models}$ est définie de façon usuelle : étant données une structure relationnelle $\mathcal{S} \in \mathcal{M}\mathrm{od}(\Sigma)$ et une valuation $\nu_{\mathcal{S}}$, on a :

- $\mathcal{S}, \nu_{\mathcal{S}} \underset{\Sigma}{\models} x = y$ si et seulement si $\nu_{\mathcal{S}}(x) = \nu_{\mathcal{S}}(y)$,
- $\mathcal{S}, \nu_{\mathcal{S}} \underset{\Sigma}{\models} \mathbf{arc}[a](x,y)$ si et seulement si $\big(\nu_{\mathcal{S}}(x), \nu_{\mathcal{S}}(y)\big) \in \mathbf{arc}[a]_{\mathcal{S}}$,
- etc.

Pour chaque formule $\varphi \in \mathcal{S}\mathrm{en}(\Sigma)$, $\mathcal{S} \underset{\Sigma}{\models} \varphi$ si et seulement si $\mathcal{S}, \nu_{\mathcal{S}} \underset{\Sigma}{\models} \varphi$ pour toute valuation $\nu_{\mathcal{S}}$. Enfin, la théorie du premier ordre associée à $\mathcal{S}$ est notée $\mathcal{T}\mathrm{h}\big[\mathcal{S}\big]$.

Soit n et p deux entiers naturels non nuls. On considère p relations n-aires quelconques $\rho_i \subseteq \prod_{j\in[n]} E_{i,j}$ où $i \in [p]$. Le produit des ρ_i est la relation n-aire

$$\prod_{i\in[p]} \rho_i \subseteq \prod_{j\in[n]} \prod_{i\in[p]} E_{i,j}$$

telle que

$$(\vec{e}_1, \ldots, \vec{e}_n) \in \prod_{i\in[p]} \rho_i \iff \forall i \in [p], \big(\pi_i(\vec{e}_1), \ldots, \pi_i(\vec{e}_n)\big) \in \rho_i \; .$$

Le *produit de p structures* $\mathcal{S}_i \in \mathcal{M}\mathrm{od}(\Sigma)$, $i \in [p]$, est la structure $\prod_{i\in[p]} \mathcal{S}_i$ de $\mathcal{M}\mathrm{od}(\Sigma)$ dont le domaine est $\prod_{i\in[p]} D_{\mathcal{S}_i}$ et qui est telle que pour tout $a \in \Sigma$, $\mathbf{arc}[a]_{\prod_{i\in[p]} \mathcal{S}_i} = \prod_{i\in[p]} \mathbf{arc}[a]_{\mathcal{S}_i}$.

Nous considérons à présent le produit de structures relationnelles par rapport à une contrainte de synchronisation. Soit p un entier naturel non nul et Σ_i, $i \in [p]$, des alphabets finis. Soit $\mathscr{C} \subseteq \prod_{i\in[p]} \widetilde{\Sigma_i}$ une contrainte de synchronisation. On définit pour tout $i \in [p]$ les injections

$$\Phi_i : \mathcal{M}\mathrm{od}(\Sigma_i) \to \mathcal{M}\mathrm{od}(\mathscr{C})$$

comme suit. Pour toute structure $\mathcal{S}_i$ dans $\mathcal{M}\mathrm{od}(\Sigma_i)$, $\Phi_i(\mathcal{S}_i)$ est la structure de $\mathcal{M}\mathrm{od}(\mathscr{C})$ dont le domaine est $D_{\mathcal{S}_i}$ et qui est telle que pour tout $(c_1, \ldots, c_p) \in \mathscr{C}$, $\mathbf{arc}[(c_1, \ldots, c_p)]_{\Phi_i(\mathcal{S}_i)} = \mathbf{arc}[c_i]_{\mathcal{S}_i}$. Le *produit des structures* $\mathcal{S}_i$ *selon la contrainte* $\mathscr{C}$ est noté $\prod^{\mathscr{C}}_{i\in[n]} \mathcal{S}_i$ et est défini par

$$\prod^{\mathscr{C}}_{i\in[n]} \mathcal{S}_i = \prod_{i\in[n]} \Phi_i(\mathcal{S}_i) \; .$$

Nous montrons maintenant que le passage dans $\mathcal{M}od(\mathscr{C})$ défini par les injections Φ_i préserve la décidabilité de la théorie du premier ordre. À cette fin, nous définissons pour tout $i \in [p]$ les applications

$$\Phi_i^{-1} : \mathcal{R}an(\Phi_i) \to \mathcal{M}od(\Sigma_i)$$

qui associent à toute structure $\mathcal{S}_i$ de $\mathcal{R}an(\Phi_i)$ l'élément $\mathcal{S}_i^{-1}$ de $\mathcal{M}od(\Sigma_i)$ tel que $\Phi_i(\mathcal{S}_i^{-1}) = \mathcal{S}_i$.

Les applications Φ_i^{-1} sont étendues à $\mathcal{S}en(\mathscr{C})$ de la façon suivante : pour tout $\varphi \in \mathcal{S}en(\mathscr{C})$, $\Phi_i^{-1}(\varphi)$ est obtenue en remplaçant dans φ les symboles $\mathbf{arc}[(c_1, \ldots, c_p)]$ par $\mathbf{arc}[c_i]$. On a alors le résultat suivant (il est bien entendu correct d'avoir $\nu_{\mathcal{S}_i}$ des deux côtés de l'équivalence car $\mathcal{S}_i$ et $\Phi_i^{-1}(\mathcal{S}_i)$ ont le même domaine).

Lemme 6.2 *Pour toute formule φ de $\mathcal{S}en(\mathscr{C})$, pour toute structure $\mathcal{S}_i$ de $\mathcal{R}an(\Phi_i)$ et pour toute valuation $\nu_{\mathcal{S}_i}$ on a :*

$$\mathcal{S}_i, \nu_{\mathcal{S}_i} \models_{\mathscr{C}} \varphi \iff \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \models_{\Sigma_i} \Phi_i^{-1}(\varphi)$$

Preuve. On montre ce lemme par récurrence sur la structure de φ.

- **Cas de base.**
  - Si φ est la formule $x = y$, alors $\Phi_i^{-1}(\varphi)$ est aussi la formule $x = y$. Comme la valuation est la même des deux côtés de l'équivalence et comme $\mathcal{S}_i$ et $\Phi_i^{-1}(\mathcal{S}_i)$ ont le même domaine, le résultat du lemme est vérifié.
  - Si φ est la formule $\mathbf{arc}[c](x, y)$, alors

$$\begin{aligned}
\mathcal{S}_i, \nu_{\mathcal{S}_i} \models_{\mathscr{C}} \varphi \quad & \Leftrightarrow \quad \big(\nu_{\mathcal{S}_i}(x), \nu_{\mathcal{S}_i}(y)\big) \in \mathbf{arc}[c]_{\mathcal{S}_i} \\
& \Leftrightarrow \quad \big(\nu_{\mathcal{S}_i}(x), \nu_{\mathcal{S}_i}(y)\big) \in \mathbf{arc}[\pi_i(c)]_{\Phi_i^{-1}(\mathcal{S}_i)} \\
& \qquad \text{car } \mathbf{arc}[\pi_i(c)]_{\Phi_i^{-1}(\mathcal{S}_i)} = \mathbf{arc}[c]_{\mathcal{S}_i} \text{ par définition de } \Phi_i \\
& \Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \models_{\Sigma_i} \mathbf{arc}[\pi_i(c)](x, y) \\
& \Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \models_{\Sigma_i} \Phi_i^{-1}(\varphi) \; .
\end{aligned}$$

- **Récurrence.** Supposons que φ_1 et φ_2 sont des formules de $\mathcal{S}en(\mathscr{C})$ pour lesquelles l'équivalence énoncée dans le lemme est vérifiée.
  - Si φ est la formule $\neg\varphi_1$, alors

$$\begin{aligned}
\mathcal{S}_i, \nu_{\mathcal{S}_i} \models_{\mathscr{C}} \varphi \quad & \Leftrightarrow \quad \mathcal{S}_i, \nu_{\mathcal{S}_i} \not\models_{\mathscr{C}} \varphi_1 \\
& \Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \not\models_{\Sigma_i} \Phi_i^{-1}(\varphi_1) \quad \text{(par hypothèse de récurrence)} \\
& \Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \models_{\Sigma_i} \neg\Phi_i^{-1}(\varphi_1) \\
& \Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \models_{\Sigma_i} \Phi_i^{-1}(\neg\varphi_1) \\
& \Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \models_{\Sigma_i} \Phi_i^{-1}(\varphi) \; .
\end{aligned}$$

– Si φ est la formule $\varphi_1 \vee \varphi_2$, alors

$$\begin{aligned}
\mathcal{S}_i, \nu_{\mathcal{S}_i} \underset{\mathscr{C}}{\models} \varphi \quad &\Leftrightarrow \quad \mathcal{S}_i, \nu_{\mathcal{S}_i} \underset{\mathscr{C}}{\models} \varphi_1 \text{ ou } \mathcal{S}_i, \nu_{\mathcal{S}_i} \underset{\mathscr{C}}{\models} \varphi_2 \\
&\Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \Phi_i^{-1}(\varphi_1) \quad \text{ou} \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \Phi_i^{-1}(\varphi_2) \\
&\qquad \text{(par hypothèse de récurrence)} \\
&\Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \Phi_i^{-1}(\varphi_1) \vee \Phi_i^{-1}(\varphi_2) \\
&\Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \Phi_i^{-1}(\varphi_1 \vee \varphi_2) \\
&\Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \Phi_i^{-1}(\varphi) \ .
\end{aligned}$$

– Si φ est la formule $\exists x \varphi_1$, alors, en notant X l'ensemble des variables qui permettent d'écrire les formules de $\mathrm{Sen}(\mathscr{C})$ et $D_{\mathcal{S}_i}$ le domaine de la structure $\mathcal{S}_i$,

$$\begin{aligned}
\mathcal{S}_i, \nu_{\mathcal{S}_i} \underset{\mathscr{C}}{\models} \varphi \quad &\Leftrightarrow \quad \text{il existe } \mu_{\mathcal{S}_i} : X \to D_{\mathcal{S}_i} \text{ telle que } \mu_{\mathcal{S}_i} \underset{X \smallsetminus \{x\}}{=} \nu_{\mathcal{S}_i} \text{ et } \mathcal{S}_i, \mu_{\mathcal{S}_i} \underset{\mathscr{C}}{\models} \varphi_1 \\
&\Leftrightarrow \quad \text{il existe } \mu_{\mathcal{S}_i} : X \to D_{\mathcal{S}_i} \text{ telle que } \mu_{\mathcal{S}_i} \underset{X \smallsetminus \{x\}}{=} \nu_{\mathcal{S}_i} \text{ et} \\
&\qquad \Phi_i^{-1}(\mathcal{S}_i), \mu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \Phi_i^{-1}(\varphi_1) \text{ (par hypothèse de récurrence)} \\
&\Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \exists x \Phi_i^{-1}(\varphi_1) \\
&\Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \Phi_i^{-1}(\exists x \varphi_1) \\
&\Leftrightarrow \quad \Phi_i^{-1}(\mathcal{S}_i), \nu_{\mathcal{S}_i} \underset{\Sigma_i}{\models} \Phi_i^{-1}(\varphi) \ .
\end{aligned}$$

□

2.2 Le résultat

Considérons p graphes $\mathcal{G}_i$, $i \in [p]$, étiquetés par $\widetilde{\Sigma_i}$ et une contrainte de synchronisation $\mathscr{C} \subseteq \prod_{i \in [p]} \widetilde{\Sigma_i}$. Pour tout $i \in [p]$, notons $|\mathcal{G}_i|$ la structure relationnelle sur l'ensemble de symboles de relations binaires $\{\mathbf{arc}[a] \mid a \in \widetilde{\Sigma_i}\}$ et telle que

– le domaine de $|\mathcal{G}_i|$ est l'ensemble des sommets V_i de $\mathcal{G}_i$ et

– pour tout a dans $\widetilde{\Sigma_i}$, $\mathbf{arc}[a]_{|\mathcal{G}_i|} = \big\{(v, v') \in V_i^2 \mid v \xrightarrow[\mathcal{G}_i]{a} v'\big\}$.

Alors, $\prod_{i \in [p]}^{\mathscr{C}} |\mathcal{G}_i| = |\prod_{i \in [p]}^{\mathscr{C}} \mathcal{G}_i|$.

D'après le lemme 6.2, pour tout $i \in [p]$, si $\mathcal{T}\mathrm{h}\big[\,|\mathcal{G}_i|\,\big]$ est décidable, $\mathcal{T}\mathrm{h}\big[\Phi_i(|\mathcal{G}_i|)\big]$ l'est également. Considérons alors le théorème de Feferman et Vaught [FV59] :

Théorème 6.3 (Feferman et Vaught) *Soit r un entier naturel et X un ensemble de variables. Soit φ une formule du premier ordre dont les variables libres appartiennent à X et dont le rang* *de quantification n'excède pas r. Alors, il existe une formule booléenne φ^b dont les variables libres sont indexées par $\Theta_r(X)$* [8] *et qui vérifie ce qui suit. Pour tout ensemble d'indices I, pour toutes structures relationnelles $\mathcal{S}_i$, $i \in I$, de domaine D_i et pour toute valuation $\nu : X \longrightarrow \prod_{i\in I} D_i$,*

$$\prod_{i\in I} \mathcal{S}_i, \nu \models \varphi \quad \textit{si et seulement si} \quad \mathcal{P}(I)^{\,9},\ \nu_I{}^{10} \models \varphi^b.$$

Rabin note dans [Rab77] que ce théorème a pour conséquence le résultat suivant :

Théorème 6.4 *Pour tout alphabet fini Σ et toutes structures $\mathcal{S}$ et $\mathcal{S}'$ de $\mathcal{M}\mathrm{od}(\Sigma)$, si $\mathcal{T}\mathrm{h}\big[\mathcal{S}\big]$ et $\mathcal{T}\mathrm{h}\big[\mathcal{S}'\big]$ sont décidables, alors $\mathcal{T}\mathrm{h}\big[\mathcal{S}\times\mathcal{S}'\big]$ l'est aussi.*

Par conséquent, si, pour tout $i \in [p]$, $\mathcal{T}\mathrm{h}\big[\,|\mathcal{G}_i|\,\big]$ est décidable, alors $\mathcal{T}\mathrm{h}\big[\,\prod^{\mathscr{C}}_{i\in[p]} |\mathcal{G}_i|\,\big]$, et donc $\mathcal{T}\mathrm{h}\big[\,|\prod^{\mathscr{C}}_{i\in[p]} \mathcal{G}_i|\,\big]$, est décidable. Pour résumer :

Théorème 6.5 *Soit $\mathcal{G}_i$, $i \in [p]$, des graphes dont la théorie du premier ordre est décidable. Soit $\mathscr{C}$ une contrainte de synchronisation sur leurs alphabets. Alors, la théorie du premier ordre de $\prod^{\mathscr{C}}_{i\in[p]} \mathcal{G}_i$ est décidable.*

3 Conséquences

Soit $\mathcal{G}$ et $\mathcal{G}'$ deux graphes préfixe-reconnaissables étiquetés par les alphabets Σ et Σ' respectivement. Supposons que $\mathcal{G}$ et $\mathcal{G}'$ ont une racine distinguée appelée r et r' respectivement. Soit $\mathscr{C} \subseteq \Sigma\times\Sigma'$ une contrainte de synchronisation. On vient de démontrer que la théorie du premier ordre du produit synchronisé usuel de $\mathcal{G}$ et $\mathcal{G}'$ selon $\mathscr{C}$ est décidable mais qu'elle devient indécidable si l'on se restreint à la partie de ce graphe accessible depuis le sommet (r,r'). Par conséquent, la théorie du premier ordre de cette partie accessible est décidable si elle est égale au produit usuel entier.

Notre objectif dans cette partie est de caractériser les graphes préfixe-reconnaissables dont le produit synchronisé usuel est égal à sa partie accessible depuis (r,r'). Nous avons relevé les faits suivants.

1. **Isomorphisme et bisimilarité.** Si $\mathcal{G}$ et $\mathcal{G}'$ sont isomorphes, alors $\mathcal{G} \underset{\mathscr{C}}{\times} \mathcal{G}'$ n'est pas forcément égal à sa partie accessible depuis (r,r') (voir l'exemple donné en figure 6.1).

8. Cet ensemble de formules est défini inductivement comme suit.
   - Soit $\Phi_{\mathrm{at}}(X)$ l'ensemble des formules atomiques du premier ordre construites sur X. À toute partie J de $\Phi_{\mathrm{at}}(X)$, on associe la formule
   $$\theta_0^J(X) = \bigwedge_{\theta\in J} \theta \wedge \bigwedge_{\theta\notin J} \neg\theta \quad \text{qui permet de poser} \quad \Theta_0(X) = \big\{\theta_0^J(X) \mid J \subseteq \Phi_{\mathrm{at}}(X)\big\}\ .$$
   - Pour tout entier naturel r et tout symbole de variable x ne figurant pas dans X on associe, à chaque partie non-vide J de $\Theta_r\big(X\cup\{x\}\big)$, la formule
   $$\theta_{r+1}^J(X) = \big(\bigwedge_{\theta\in J} \exists x\theta\big) \wedge \big(\forall x \bigvee_{\theta\in J} \theta\big) \text{ qui permet de poser } \Theta_{r+1}(X) = \big\{\theta_{r+1}^J(X) \mid \varnothing \subsetneq J \subseteq \Theta_r\big(X\cup\{x\}\big)\big\}\ .$$

9. $\mathcal{P}(I)$ désigne l'algèbre de Boole de l'ensemble des parties de I.

10. La valuation ν_I est à valeurs dans $\mathcal{P}(I)$ et est définie comme suit. Pour toute formule $\theta \in \Theta_r(X)$ et toute variable x_θ indexée par θ, $\nu_I(x_\theta)$ est l'ensemble $\{i \in I \mid \mathcal{S}_i, \nu_i \models \theta\}$ (en notant ν_i la valuation telle que, pour tout $y \in X$, $\nu_i(y) = \pi_i(\nu(y))$).

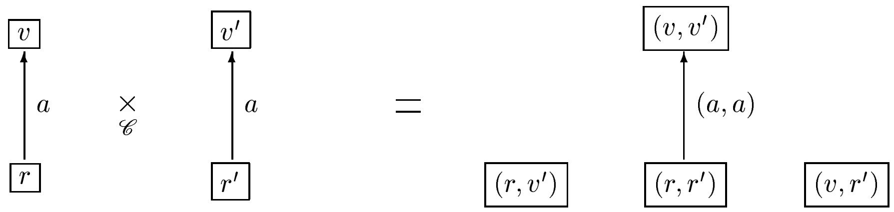


FIG. 6.1 – *Produit de deux graphes isomorphes selon la contrainte* $\mathscr{C} = \{(a, a)\}$.

Il y a également des cas où le produit synchronisé de deux graphes qui ne sont pas bisimilaires donne un graphe égal à sa partie accessible depuis (r, r') (voir l'exemple donné en figure 6.2).

2. La **condition** suivante est **suffisante**. Si r et r' portent une ε-boucle et si $\mathscr{C} = \widetilde{\Sigma} \times \widetilde{\Sigma}'$, alors le produit synchronisé usuel selon $\mathscr{C}$ est égal à sa partie accessible à partir de (r, r'). En effet, soit v un sommet de $\mathcal{G}$ et v' un sommet de $\mathcal{G}'$. Soit c un chemin de $\mathcal{G}$ menant à v et c' un sommet de $\mathcal{G}'$ menant à v'. Supposons que la longueur de c, notée $|c|$, est plus petite que celle de c', notée $|c'|$. Alors, pour aller à (v, v'), on réalise $|c'| - |c|$ fois l'opération « boucler sur r et progresser sur un arc de c' » puis on réalise l'opération « progresser dans c et c' jusqu'à parvenir à v et v' ».

3. Les deux **conditions** suivantes sont **nécessaires**.

   (a) Si $\mathcal{G}$ a au moins deux sommets distincts, alors r' a un degré entrant non nul et si $\mathcal{G}'$ a au moins deux sommets distincts, alors r a un degré entrant non nul. En effet, si v (resp. v') est un sommet de $\mathcal{G}$ (resp. $\mathcal{G}'$) distinct de r (resp. r'), pour que le sommet (v, r') (resp. (r, v')) soit accessible, il faut qu'il existe un chemin non-vide (*i.e.* comptant au moins un arc) de r' à r' (resp. de r à r) dans $\mathcal{G}'$ (resp. dans $\mathcal{G}$).

   (b) Pour tout sommet v de $\mathcal{G}$ et pour tout sommet v' de $\mathcal{G}'$ tels que $(v, v') \neq (r, r')$, en notant $\to^{-1}(v)$ (resp. $\to^{-1}(v')$) l'ensemble des étiquettes des arcs qui arrivent sur v (resp. sur v') on a

$$\left(\to^{-1}(v) \times \to^{-1}(v')\right) \cap \mathscr{C} \neq \varnothing\ .$$

   Ces conditions ne sont pas suffisantes. Il suffit de considérer l'exemple de la figure 6.3 (circuits non-bisimilaires) et celui de la figure 6.4 (circuits isomorphes). Dans ces deux cas, les graphes vérifient le point 3b ci-dessus et leur racine distinguée a un degré entrant non-nul.

4. Dans le cas où $\mathcal{G}$ et $\mathcal{G}'$ sont tels que

   (a) r et r' portent une ou plusieurs boucles et n'ont pas d'autre arc entrant que ces boucles,

   (b) $\mathcal{G}$ et $\mathcal{G}'$ ne contiennent pas de circuits autres que les boucles portées par leur racine distinguée et

   (c) en inversant le sens de tous les arcs, on obtient des graphes tels que tous les chemins qui partent d'un sommet quelconque mènent à la racine distinguée,

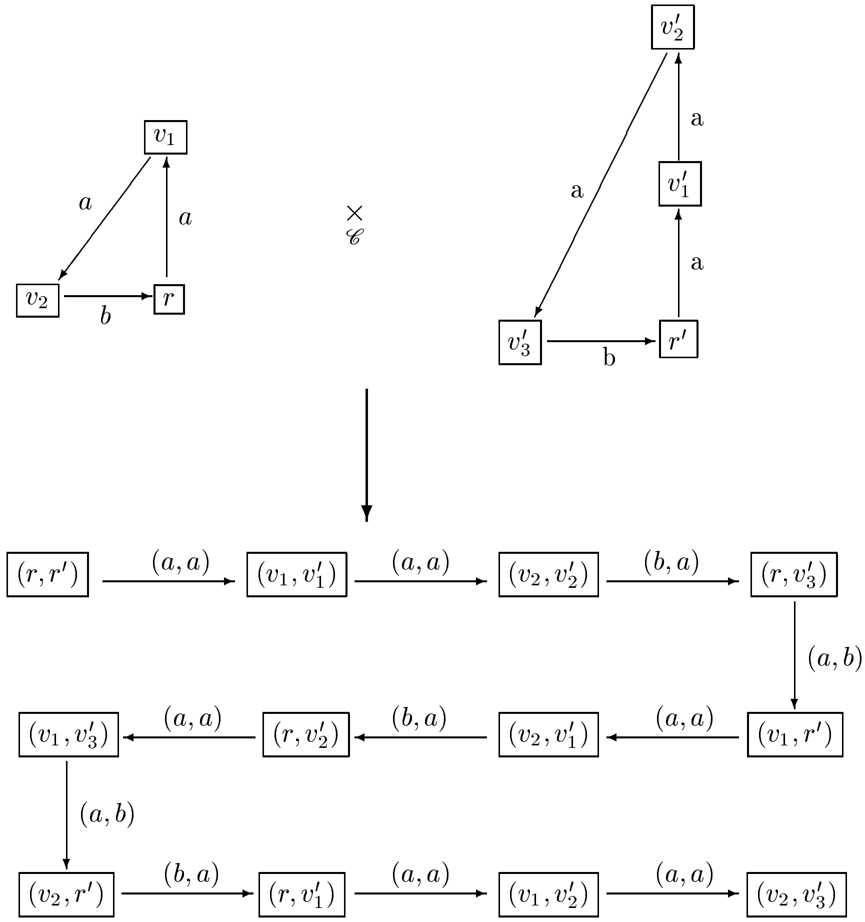


FIG. 6.2 – *Produit de deux graphes non-bisimilaires selon la contrainte* $\mathscr{C} = \{(a,a),(a,b),(b,a)\}$.

la condition 3b est suffisante.

En effet, supposons que cette condition soit vérifiée. Soit v un sommet de $\mathcal{G}$ et v' un sommet de $\mathcal{G}'$. Alors, si $(v,v') \neq (r,r')$, il existe v_1 dans $\mathcal{G}$ et v'_1 dans $\mathcal{G}'$ tels que $v_1 \xrightarrow{a_1} v$ et $v'_1 \xrightarrow{a'_1} v'$ et $(a_1,a'_1) \in \mathscr{C}$. Si $(v_1,v'_1) \neq (r,r')$, il existe aussi v_2 dans $\mathcal{G}$ et v'_2 dans $\mathcal{G}'$ tels que $v_2 \xrightarrow{a_2} v_1$ et $v'_2 \xrightarrow{a'_2} v'_2$ et $(a_2,a'_2) \in \mathscr{C}$, etc. Vu les conditions 4b et 4c, en continuant ainsi on arrive à un entier naturel n tel que $v_n = r$ ou bien $v'_n = r'$. Si $v_n = r$ et $v'_n \neq r'$, en bouclant sur r (ce qui est possible d'après 4a) et en continuant à progresser dans $\mathcal{G}'$, on arrive à un m tel que $v_n = r$ et $v'_n = r'$. Donc (v,v') est accessible. Le cas $v'_n = r'$ est identique.

L'importance de la condition 4c imposée à $\mathcal{G}$ et $\mathcal{G}'$ est illustrée par l'exemple suivant. On considère le graphe $\mathcal{G}$ de la figure 6.5. Si l'on inverse le sens de tous les arcs de $\mathcal{G}$, à partir de tout sommet de la forme a^ib il existe un chemin infini étiqueté par un mot qui ne contient que des c qui ne passe pas par la racine distinguée. Le produit synchronisé $\mathcal{G} \underset{\mathscr{C}}{\times} \mathcal{G}$ selon la contrainte

$$\mathscr{C} = \{a,b,c,\varepsilon\}^2 \setminus \{(b,b)\}$$

est tel que ses sommets de la forme (a^ib, a^ib) ne sont pas accessibles depuis $(\varepsilon,\varepsilon)$. Pourtant, la condition 3b est bien vérifiée.

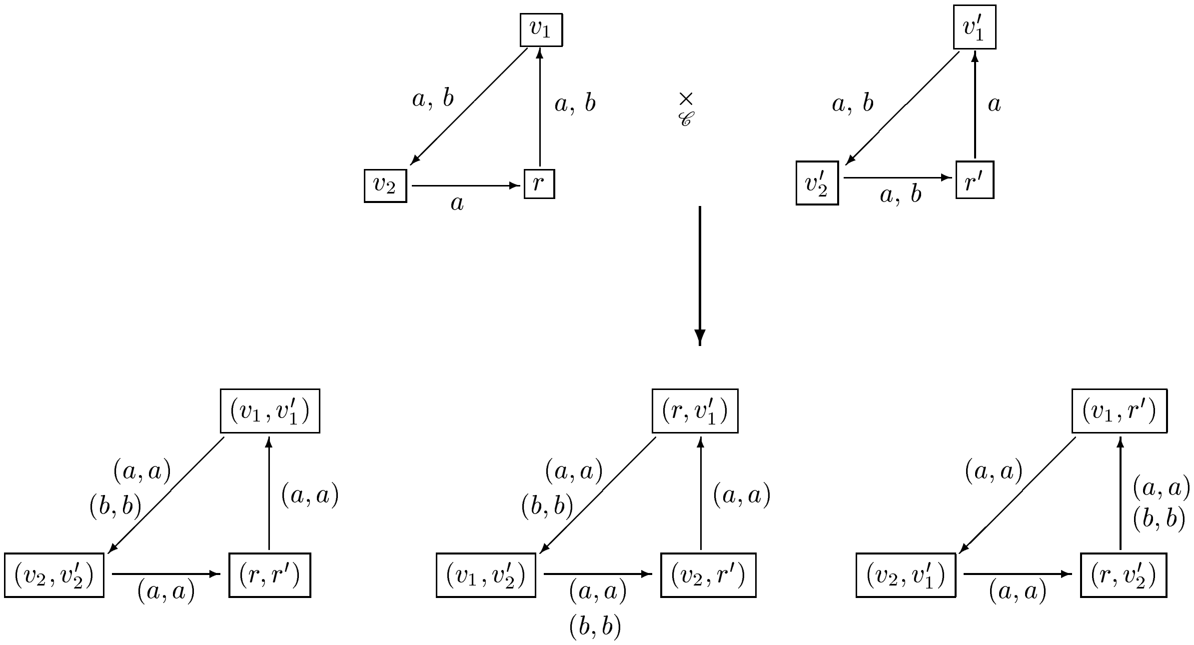


FIG. 6.3 – *Produit de deux circuits non-bisimilaires selon la contrainte* $\mathscr{C} = \{(a,a),(b,b)\}$.

Enfin, notons que tout **arbre** vérifie les conditions 4b et 4c. Par conséquent, si $\mathcal{G}$ et $\mathcal{G}'$ vérifient 4a et si en leur retirant les boucles portées par leur racine distinguée ils deviennent des arbres, alors la condition 3b est nécessaire et suffisante pour que tous les états de leur produit synchronisé usuel soient accessibles depuis (r, r').

5. Dans le cas où l'un des deux graphes est un circuit et l'autre vérifie les points 4a, 4b et 4c, la condition 3b est également suffisante. En effet, le raisonnement précédent est valable dans ce cas aussi.

6. Dans le cas où $\mathcal{G}$ et $\mathcal{G}'$ sont des **circuits**, on peut faire les remarques suivantes. On appelle *chemin simple d'un circuit* tout chemin comptant au moins un arc et dont les arcs sont tous distincts. Dans un même circuit, tous les chemins simples allant d'un sommet donné v à un autre v' ont le même nombre d'arcs. Ce nombre est appelé *éloignement* de v à v' et est noté $e(v, v')$. Enfin, pour tous sommets v et v' d'un circuit, les nombres $e(v, v)$ et $e(v', v')$ sont égaux et définissent la *longueur* du circuit.

   Soit n la longueur de $\mathcal{G}$ et n' celle de $\mathcal{G}'$. On considère deux cas :

   (a) Celui où n et n' sont premiers entre eux. Alors, la condition 3b est suffisante car pour tout sommet v de $\mathcal{G}$ et tout sommet v' de $\mathcal{G}'$ on peut trouver un chemin de $\mathcal{G}$ allant de r à v et un chemin de $\mathcal{G}'$ allant de r' à v' ayant tous les deux le même nombre d'arc.

   En effet, tout chemin de $\mathcal{G}$ allant de r à v est tel qu'il existe $k \in \mathbb{N}$ tel que son nombre d'arcs est $e(r, v) + n.k$. De même, tout chemin de $\mathcal{G}'$ allant de r' à v' est tel qu'il existe $k' \in \mathbb{N}$ tel que son nombre d'arcs est $e(r', v') + n'.k'$.

   Comme n et n' sont premiers entre eux, d'après le théorème de Bezout il existe deux entiers relatifs m et m' qui sont tels que $n.m + n'.m' = 1$, *i.e.*, en multipliant les deux

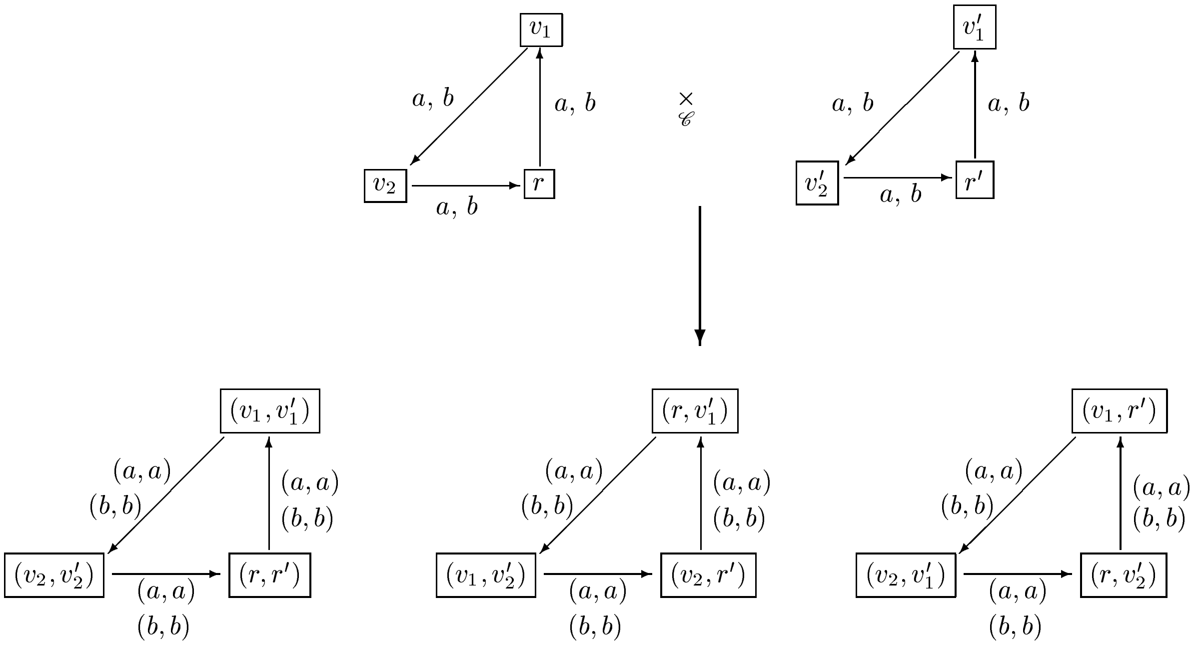


FIG. 6.4 – *Produit de deux circuits isomorphes selon la contrainte* $\mathscr{C} = \{(a,a),(b,b)\}$.

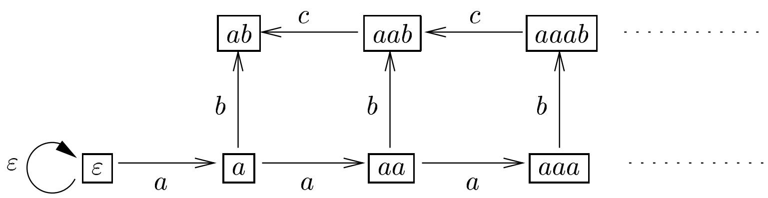


FIG. 6.5 – *Un graphe qui ne vérifie pas la condition 4c.*

membres de cette égalité par $-e(r,v) + e(r',v')$, qui sont tels que

$$n.\Big[m.\big(-e(r,v) + e(r',v')\big)\Big] + n'.\Big[m'.\big(-e(r,v) + e(r',v')\big)\Big] = -e(r,v) + e(r',v') \ .$$

Si l'on pose $k = m.\big(-e(r,v) + e(r',v')\big)$ et $k' = m'.\big(e(r,v) - e(r',v')\big)$, cette égalité devient $n.k - n'.k' = -e(r,v) + e(r',v')$, ce qui est équivalent à

$$e(r,v) + n.k = e(r',v') + n'.k' \ .$$

Par conséquent, il suffit de prendre k et k' comme indiqué ci-dessus pour obtenir un chemin de $\mathcal{G}$ allant de r à v et un chemin de $\mathcal{G}'$ allant de r' à v' ayant tous les deux le même nombre d'arcs.

(b) Celui où n' est un multiple de n. Dans ce cas, la démonstration précédente ne marche pas car si $n' = a.n$, on a

$$\begin{aligned} & e(r,v) + n.k = e(r',v') + n'.k' \\ \Longleftrightarrow \quad & e(r,v) - e(r',v') = a.n.k' - n.k \\ \Longleftrightarrow \quad & \tfrac{e(r,v)-e(r',v')}{n} = a.k' - k \ . \end{aligned}$$

Par conséquent, pour les sommets v de $\mathcal{G}$ et v' de $\mathcal{G}'$ qui sont tels que $\frac{e(r,v)-e(r',v')}{n}$ n'est pas un entier, il n'existe aucun chemin $P(r,v)$ de $\mathcal{G}$ allant de r à v et aucun chemin $P'(r',v')$ de $\mathcal{G}'$ allant de r' à v' tels que $P(r,v)$ et $P'(r',v')$ ont le même nombre d'arcs. Ces sommets v et v' sont tels que (v,v') n'est pas accessible dans le produit synchronisé de $\mathcal{G}$ et $\mathcal{G}'$.

Conclusion

Comme nous l'avons explicité dans l'introduction, les travaux présentés dans cette thèse ont pour cadre la vérification formelle des systèmes informatiques distribués, concurrents ou réactifs qui sont aujourd'hui la clé du fonctionnement et de l'efficacité de nombreux secteurs d'activité. Cette vérification est basée sur un modèle mathématique du système à vérifier, le plus souvent un système de transitions étiquetées. Or, dans la plupart des cas réalistes, les systèmes de transitions obtenus ont un nombre d'états immense ce qui pose de sérieux problèmes à leur manipulation informatique.

Une technique qui permettrait de remédier au problème du stockage des états des systèmes de transitions est l'utilisation de graphes infinis décrits au moyen d'une structure finie et compacte. De cette façon, un système de transitions fini très grand mais présentant une forte régularité pourrait être manipulé par le biais de la description d'un graphe infini qui serait une approximation du système.

L'étude des graphes infinis a permis de dégager un certain nombre de classes présentant des propriétés intéressantes. Pour chacune de ces classes, plusieurs descriptions finies ont déjà été mises en évidence. Nous avons détaillé au chapitre 3 quelques descriptions connues des éléments de certaines sous-classes des graphes rationnels :

- pour ce qui est des graphes *préfixe-reconnaissables*, nous avons explicité les relations préfixe-reconnaissables, les transformations de l'arbre binaire complet au moyen d'une substitution rationnelle inverse suivie d'une restriction rationnelle et enfin les systèmes d'équations régulières faisant intervenir des opérations de remplacement de sommets ;

- pour ce qui est des graphes *réguliers*, nous avons présenté les grammaires déterministes de graphes et les systèmes d'équations régulières faisant intervenir des opérations de remplacement d'hyperarcs ;

- enfin, nous avons présenté les *machines à pile* et les graphes qui leur sont associés.

Bien que la classe des graphes rationnels soit encore peu connue, on sait déjà qu'elle est fermée par produit synchronisé, ce qui ouvre des perspectives encourageantes à la spécification de systèmes de processus communiquants au moyen de ses graphes. Cependant, pour ce qui est du problème de la vérification sur les graphes rationnels, un résultat récent [Mor00] a montré que même la théorie du premier ordre de ces graphes n'est pas décidable. Néanmoins, les graphes préfixe-reconnaissables, les graphes réguliers et les graphes associés aux machines à pile sont tous rationnels et possèdent la propriété intéressante d'avoir une théorie monadique du second ordre qui est décidable. Dans le cas des graphes préfixe-reconnaissables, ce résultat a été démontré en ne considérant pas la quantification sur les arcs. Dans le cas des graphes réguliers et des graphes des machines à pile, ce résultat a été démontré pour la logique monadique du second ordre avec comptage modulaire et quantification sur les arcs. Malheureusement, aucune des trois classes n'est fermée par produit synchronisé, ce qui limite sérieusement les possibilités de spécification de systèmes de processus communiquants au moyen de leurs éléments.

Résumé du travail effectué

L'objectif de notre travail était l'étude, dans le cadre de la vérification formelle, des graphes associés à des machines de Turing et à des spécifications de Thue. Notre motivation était d'une part de mettre en évidence un certain nombre de propriétés des classes de graphes liées aux langages sensibles au contexte et aux langages récursivement énumérables car ces classes n'ont bénéficié d'aucune étude approfondie. D'autre part, nous souhaitions examiner les graphes des spécifications de Thue ; le principal attrait de ces dernières est qu'elles permettent d'exprimer des propriétés dynamiques de manière syntaxique en terme de la théorie des langages, ce qui laisse envisager l'utilisation des techniques de démonstration automatique et de réécriture de mots pour vérifier ces propriétés.

Pour ce qui est des spécifications de Thue, nous avons détaillé une construction qui montre que la classe de leurs graphes est fermée par produit synchronisé. À partir de n spécifications de Thue $\mathcal{T}_i = \big(\Omega_i, \Delta_i, \mathcal{S}_i, R_i, u_i\big)$ et d'une contrainte de synchronisation $\mathscr{C} \subseteq \prod_{i\in[n]} \Delta_i$, nous avons construit une spécification de Thue

$$\prod_{i\in[n]}^{\mathscr{C}} \mathcal{T}_i = \big(\Omega, \Delta, \mathcal{S}, R, u\big)$$

dont le graphe est isomorphe au produit synchronisé de ceux des $\mathcal{T}_i$ selon $\mathscr{C}$. Cette spécification est telle que

- le semi-système de Thue $\mathcal{S}$ est composé de règles qui s'appliquent à des mots du langage $\big[\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\big]^*$ et il est constitué d'un ensemble convergent de règles de réarrangement qui permettent de « normaliser » des mots et d'un ensemble de règles de simulation qui permettent d'appliquer les règles des $\mathcal{S}_i$,
- le langage rationnel R est défini comme étant celui reconnu par un produit d'automates reconnaissant les R_i et
- le mot u est la « normalisation » du vecteur $(u_1, \ldots, u_n)$.

Cette construction est cependant exponentielle en le nombre n. Nous l'avons montré en fournissant des formules qui permettent de calculer précisément le nombre de règles de $\mathcal{S}$ et la taille de l'automate fini que l'on a construit et qui reconnaît R.

Enfin, nous avons montré que la classe des graphes rationnels sans ε-arc est incluse, à isomorphisme près, dans celle des graphes des spécifications de Thue ; nous en avons déduit l'indécidabilité de la théorie du premier ordre des graphes des spécifications de Thue.

En ce qui concerne les graphes des machines de Turing, nous avons démontré un certain nombre de résultats.

- Nous avons examiné en premier lieu le problème de l'association d'une machine à une seule bande de travail à une machine comptant plusieurs de ces bandes de telle sorte que le graphe de la machine de départ soit ε-équivalent à celui de celle associée. Nous avons fourni une construction qui permet de réaliser une telle association et qui produit une machine linéairement bornée si celle de départ est linéairement bornée.

- Dans un second temps, nous avons étudié la fermeture de la classe des graphes des machines de Turing par produit synchronisé. Nous avons montré que cette classe est fermée, à isomorphisme près, par cette opération. Si l'on considère la classe des machines à une seule bande de travail, le résultat montré en premier lieu permet de prouver qu'elle est fermée, à ε-équivalence près, par produit synchronisé. Par ailleurs, les arguments exposés au cours de cette démonstration restent valides dans le cas de machines linéairement bornées. Nous avons donc également signalé que la classe des graphes de ces machines est fermée, à isomorphisme près, par produit synchronisé.
- Nous avons ensuite considéré les liens qui unissent les graphes des machines de Turing à ceux des spécifications de Thue.
  - Nous avons construit à partir d'une spécification de Thue quelconque une machine de Turing dont le graphe est ε-équivalent à celui de la spécification. Dans le cas où la spécification dont on part est linéaire, *i.e.* il existe une fonction linéaire f telle que toute réécriture d'un mot w par les règles de son semi-système de Thue aboutit à un mot dont la longueur de dépasse pas $f(|w|)$, la machine de Turing construite est linéairement bornée. Nous avons alors noté, en considérant un exemple très simple, qu'une telle construction n'était pas toujours possible dans le sens inverse car d'une part le graphe de toute spécification de Thue ne contient aucun arc étiqueté par le mot vide ε et d'autre part il existe des graphes de machines de Turing qui ne sont ε-équivalents à aucun graphe sans ε-arc.
  - À partir de toute machine de Turing $\mathcal{M}$ dont le graphe vérifie trois conditions bien précises, nous avons construit une spécification de Thue dont le graphe est ε-équivalent à celui de la machine. Ces conditions portent essentiellement sur les éventuels ε-chemins qui existent dans le graphe de $\mathcal{M}$. Nous avons signalé que toute machine de Turing temps-réel, *i.e.* dont le graphe ne contient aucun ε-arc, vérifie toujours les trois conditions énoncées et que la spécification de Thue construite dans ce cas est longueur décroissante (*i.e.* toute réécriture d'un mot w par les règles de son semi-système de Thue aboutit à un mot dont la longueur de dépasse pas celle de w).
- Finalement, nous avons étudié le problème de la conservation de la décidabilité de la théorie du premier ordre de graphes par produit synchronisé. Nous avons examiné deux cas, celui où les graphes considérés n'ont pas forcément une racine et celui où les graphes ont une racine distinguée et où leur produit synchronisé est la restriction du produit usuel à sa partie accessible depuis le vecteur composé des racines. Pour ce dernier cas, nous avons montré que même en considérant les graphes des machines à pile, *i.e.* les graphes se situant au niveau le plus bas de la hiérarchie

  $$\{\text{graphes des machines à pile}\} \subsetneq \{\text{graphes réguliers}\} \subsetneq \{\text{graphes préfixe-reconnaissables}\}\ ,$$

  la théorie du premier ordre du produit synchronisé n'est pas décidable. Par contre, dans le cas de graphes sans racine distinguée et du produit synchronisé usuel, nous avons obtenu, grâce à une conséquence d'un théorème démontré par Solomon Feferman et Robert Vaught, le résultat intéressant suivant : si la théorie du premier ordre des graphes dont on part est décidable, alors la théorie du premier ordre de leur produit synchronisé est décidable.

Nous présentons à la fin de cette conclusion un tableau synthétique qui regroupe les principaux résultats concernant le produit synchronisé et la décidabilité de théories des graphes dont il a été question dans cette thèse.

Perspectives

Les travaux présentés dans cette thèse demandent, à notre sens, à être prolongés sur les plans suivants.

- La connaissance des liens qui unissent les graphes des spécifications de Thue à ceux des machines de Turing mériterait d'être développée. Nous avons prouvé au chapitre 5 que la classe des graphes des spécifications de Thue est ε-équivalente à un sous-ensemble strict de celle des graphes des machines de Turing. Nous avons également mis en évidence trois conditions suffisantes assurant que le graphe de toute machine de Turing les vérifiant est ε-équivalent à celui d'une spécification de Thue. Nous avons montré que ces conditions ne sont pas nécessaires, ce qui nous a conduit à une conjecture fournissant une condition suffisante plus puissante que les trois que nous avons considérées en premier lieu. Cette conjecture demande à être examinée plus en détail.

- Les graphes que nous avons associés aux machines de Turing sont constitués des configurations internes accessibles depuis une configuration initiale ainsi que des arcs qui les relient. C'est la théorie du premier ordre de ces graphes que nous avons considérée au chapitre 5 de ce document.

  D'un point de vue théorique, il serait intéressant d'examiner les graphes associés de la même façon aux machines de Turing mais en considérant toutes les configurations internes (y compris celles qui ne sont pas accessibles depuis la configuration initiale) et les arcs qui les relient. La question serait alors de savoir si la théorie du premier ordre de ces graphes est décidable.

- Nous avons montré au chapitre 6 que si la théorie du premier ordre de graphes donnés est décidable, alors la théorie du premier ordre de leur produit synchronisé est également décidable. La preuve de ce résultat utilise une conséquence du théorème de Feferman et Vaught. La démonstration de ce théorème consiste, étant données n structures relationnelles S_i, une valuation ν et une formule du premier ordre φ, à construire une valuation ν' et une formule booléenne φ^b telle que

  $$\Big(\prod_{i\in[n]} S_i\Big), \nu_i \models \varphi \iff \mathcal{P}\big([n]\big), \nu' \models \varphi^b$$

  (le symbole $\mathcal{P}\big([n]\big)$ désignant l'algèbre de Boole de l'ensemble des parties de $[n]$). Cependant, la construction de la formule φ^b semble être, au premier abord, d'une complexité non-élémentaire et mériterait un examen approfondi.

Classe de graphes	**Représentations**	**Décidabilité de théories**	**Fermée par produit**	**Représentation du produit**	**Décidabilité de théories du produit**
Graphes des machines à pile	Machines à pile	La théorie monadique du second ordre avec comptage modulaire et quantification sur les arcs est décidable.	Non	Graphes rationnels	La théorie du premier ordre du produit entier est décidable. La théorie du premier ordre de la partie accessible du produit n'est pas décidable.
Graphes réguliers	Grammaires déterministes de graphes Systèmes d'équations régulières avec opérations de remplacement d'hyperarcs				La théorie du premier ordre du produit entier est décidable. Dans le cas de graphes avec une racine distinguée, la théorie du premier ordre de la partie accessible du produit n'est pas décidable.
Graphes préfixe-reconnaissables	Relations préfixe-reconnaissables Transformations de l'arbre binaire complet Systèmes d'équations régulières avec opérations de remplacement de sommets	La théorie monadique du second ordre sans quantification sur les arcs est décidable.			
Graphes rationnels	Relations rationnelles	La théorie du premier ordre n'est pas décidable.	oui		La théorie du premier ordre du produit entier n'est pas décidable. Dans le cas de graphes avec une racine distinguée, la théorie du premier ordre de la partie accessible du produit n'est pas décidable.
Graphes des Spécifications de Thue	Spécifications de Thue			Spécifications de Thue	
Graphes de machines linéairement bornées	Machines linéairement bornées			Machines linéairement bornées	La théorie du premier ordre de la partie accessible du produit n'est pas décidable.
Graphes des machines de Turing	Machines de Turing			Machines de Turing	

Résumé des notations

Notations mathématiques

- $\mathbb{N}$ est l'ensemble des entiers naturels,
- $[n]$ (où $n \in \mathbb{N}$) est l'ensemble $\{1, \ldots, n\}$ (avec $[0] = \varnothing$),
- $\mathcal{D}\mathrm{om}(\mathbf{R})$ (où $\mathbf{R}$ est une relation binaire) est le domaine de $\mathbf{R}$,
- $\mathcal{R}\mathrm{an}(\mathbf{R})$ (où $\mathbf{R}$ est une relation binaire) est l'image de $\mathbf{R}$,
- $\pi_i(\vec{t})$ (où $i \in [n]$ et $\vec{t}$ est un n-uplet) est la i^{e} coordonnée de $\vec{t}$,

Théorie des langages

- Σ est en général un alphabet fini,
- Σ^* est le monoïde libre engendré par Σ,
- ε est le mot vide,
- Σ^+ est l'ensemble $\Sigma^* \setminus \{\varepsilon\}$
- $\widetilde{\Sigma}$ est l'ensemble $\Sigma \cup \{\varepsilon\}$
- $|\Sigma|$ est le nombre de caractères de Σ,
- $|w|$ (où w est un mot construit sur Σ) est le nombre de caractères de w,
- $\mathcal{S}\mathrm{uff}(w)$ est l'ensemble des suffixes du mot w,
- $\mathcal{S}\mathrm{uff}^+(w)$ est l'ensemble des suffixes non-vides du mot w.

Graphes

- $v \xrightarrow[\mathcal{G}]{w} v'$ est un chemin étiqueté par w dans le graphe $\mathcal{G}$ allant du sommet v au sommet v' ;
- $\overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}}$ est la fermeture réflexive et transitive de $\xrightarrow[\mathcal{G}]{\varepsilon}$;

- $\overset{a}{\underset{\mathcal{G}}{\dashrightarrow}} = \overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}} \circ \overset{a}{\underset{\mathcal{G}}{\longrightarrow}} \circ \overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}}$;
- $(\mathcal{G}, r)$ signifie que $\mathcal{G}$ est un graphe dont r est une racine que l'on distingue.

Classes de graphes

- $\mathcal{G}[\text{TS}]$ est la classe des graphes des spécifications de Thue,
- $\mathcal{G}[\text{Lin-TS}]$ est la classe des graphes des spécifications de Thue linéaires,
- $\mathcal{G}[\text{Len-TS}]$ est la classe des graphes des spécifications de Thue longueur-décroissante,
- $\mathcal{G}[\text{TM}]$ est la classe des graphes des machines de Turing,
- $\mathcal{G}[\text{Lin-TM}]$ est la classe des graphes des machines de Turing linéairement bornées,
- $\mathcal{G}[\text{Real-TM}]$ est la classe des graphes des machines de Turing temps-réel.

Bibliographie

[AD90] Rajeev ALUR et David L. DILL. – Automata for modelling real-time systems. *In : Proc. 17th Inter. Col. on Automata, Languages and Programming, ICALP90, Warwick University, England*, *Lecture Notes in Computer Science*, volume 443. pp. 322–335. – Springer Verlag, 1990.

[AK79] Jirí ADÁMEK et Václav KOUBEK. – Least fixed point of a functor. *Journal of Computer and System Sciences*, n° 19, 1979, pp. 163–178.

[AN82] André ARNOLD et Maurice NIVAT. – Comportements de processus. *In : Colloque AFCET "Les mathématiques de l'Informatique"*, pp. 35–68. – 1982.

[AR91] Egidio ASTESIANO et Gianna REGGIO. – Algebraic specification of concurrency. *In : Recent Trends in Data Type Specification*, éd. par Michel BIDOIT et Christine CHOPPY, *Lecture Notes in Computer Science*, volume 655, pp. 1–39. – Dourdan, septembre 1991. Selected papers from the 8^{th} Workshop on Specification of Abstract Data Types.

[Arn92] André ARNOLD. – *Systèmes de transitions finis et sémantique des procesus communicants.* – Masson, 1992, *Études et recherches en informatique.*

[AS85] Bowen ALPERN et Fred B. SCHNEIDER. – *Defining safety and liveness.* – Rapport de recherche, Cornell Univ., Ithaca, NY, 1985.

[AS87] Bowen ALPERN et Fred B. SCHNEIDER. – Recognizing safety and liveness. *Distributed Comput.*, n° 2, 1987, pp. 117–126.

[Bar97] Klaus BARTHELMANN. – *On Equational Simple Graphs.* – Rapport de recherche n° 9/97, Mainz, Johannes Gutenberg Universität, 1997.

[Bar98] Klaus BARTHELMANN. – When can an equational simple graph be generated by hyperedge replacement. *In : Proc. 3rd Inter. Symp. on Mathematical Foundations of Computer Science, Brno, Czech Republic*, éd. par L. BRIM, J. GRUSKA et J. ZLATUSKA, *Lecture Notes in Computer Science*, volume 1450, pp. 543–552. – août 1998.

[Bau91] Michel BAUDERON. – Infinite hypergraphs I. Basic properties. *Theoretical Computer Science*, n° 82, 1991, pp. 177–214.

[Bau92] Michel BAUDERON. – Infinite hypergraphs II. Systems of recursive equations. *Theoretical Computer Science*, n° 103, 1992, pp. 165–190.

[Büc60] J. Richard BÜCHI. – On a decision method in restricted second-order arithmetic. *In : Proc. Inter. Congr. on Logic, Methodology and Philosophy of Science*, éd. par E. NAGEL. pp. 1–11. – Stanford Univ. Press, Stanford, CA, 1960.

[BCM+90] Jerry R. BURCH, Edmund M. CLARKE, Kenneth L. MCMILLAN, David L. DILL et L. J. HWANG. – Symbolic model checking : 10^{20} states and beyond. *In : Proc. 5th Ann. Symp. on Logic in Computer Science*. – juin 1990.

[BJM97] Adel BOUHOULA, Jean Pierre JOUANNAUD et José MESEGUER. – Specification and proof in membership equational logic. *In : Invited talk at the Inter. Col. on Trees in Algebra and Programming - Proc. of TAPSOFT'97, Lille, France*, éd. par M. BIDOIT et M. DAUCHET, *Lecture Notes in Computer Science*, volume 1214, pp. 67–92. – avril 1997.

[BKP84] Howard BARRINGER, Ruurd KUIPER et Amir PNUELI. – Now you may compose temporal logic specifications. *In : Proc. 16th Ann. ACM Symp. on Theory of Computing*, pp. 51–63. – 1984.

[BKP86] Howard BARRINGER, Ruurd KUIPER et Amir PNUELI. – A really abstract concurrent model and its temporal logic. *In : Proc. 13th Ann. ACM Symp. on Principles of Programming Languages*, pp. 173–183. – 1986.

[Bry86] Randal E. BRYANT. – Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, vol. C-35, n° 8, 1986.

[BS92] Olaf BURKART et Bernhard STEFFEN. – Model checking for context-free processes. *In : Proc. 3rd Inter. Conf. on Concurrency Theory, CONCUR'92, Stony Brook, NY, USA*, éd. par R. CLEAVELAND, *Lecture Notes in Computer Science*, volume 630. pp. 123–137. – Springer, août 1992.

[BS94] Olaf BURKART et Bernhard STEFFEN. – Pushdown processes : parallel composition and model checking. *In : Proc. 5th Inter. Conf. on Concurrency Theory, CONCUR'94, Uppsala, Sweden*, éd. par B. JONSSON et J. PARROW, *Lecture Notes in Computer Science*, volume 836. pp. 98–113. – Springer, août 1994.

[Cau92] Didier CAUCAL. – On the regular structure of prefix rewriting. *Theoretical Computer Science*, n° 106, 1992, pp. 61–86.

[Cau95] Didier CAUCAL. – Bisimulation of context-free grammars and of pushdown automata. *In : Modal logic and process algebra, CSLI*, volume 53, pp. 85–106. – Stanford, 1995.

[Cau96] Didier CAUCAL. – On infinite transition graphs having a decidable monadic second–order theory. *In : Proc. 23rd Inter. Col. on Automata Languages and Programming, Paderborn, Germany*, éd. par F. Meyer auf der HEIDE et B. MONIEN, *Lecture Notes in Computer Science*, volume 1099, pp. 194–205. – 1996.

[CE81] Edmund M. CLARKE et E. Allen EMERSON. – Design and synthesis of synchronization skeletons using Branching Time Temporal Logic. *In : Proc. Workshop on Logics of Programs*, *Lecture Notes in Computer Science*, volume 131. pp. 52–71. – Springer Verlag, Berlin, 1981.

[CK98] Hugues CALBRIX et Teodor KNAPIK. – A string-rewriting characterization of context-free graphs. *In : Proc. 18th Inter. Conf. on Foundations of Software Technology and Theoretical Computer Science, Chennai, India*, éd. par V. ARVIND et R. RAMANUJAM, *Lecture Notes in Computer Science*, volume 1530. pp. 331–34. – Springer Verlag, 1998.

[CK99a] Didier CAUCAL et Teodor KNAPIK. – *On Internal Presentation of Regular Graphs.* – Rapport de recherche n° INF/99/07/02/a, Université de la Réunion, juillet 1999.

[CK99b] Didier CAUCAL et Teodor KNAPIK. – A string-rewriting characterization of recognizable graphs. – 1999. Unpublished draft.

[CLM89] Edmund M. CLARKE, David E. LONG et Kenneth L. MCMILLAN. – Compositional model checking. *In : Proc. 4th Ann. IEEE Symp. on Logic in Computer Science.* pp. 353–362. – IEEE Computer Society Press, 1989.

[Cou89] Bruno COURCELLE. – The monadic second–order logic of graphs, II : Infinite graphs of bounded width. *Mathematical Systems Theory*, vol. 21, 1989, pp. 187–221.

[Cou90a] Bruno COURCELLE. – Graph rewriting : An algebraic and logic approach. *In : Formal Models and Semantics*, éd. par Jan van LEEUWEN, *Handbook of Theoretical Computer Science*, volume B, pp. 193–242. – Elsevier, 1990.

[Cou90b] Bruno COURCELLE. – The monadic second–order logic of graphs, I : Recognizable sets of finite graphs. *Information and Computation*, vol. 85, 1990, pp. 12–75.

[Cou94] Bruno COURCELLE. – The monadic second–order logic of graphs, VI : On several representations of graphs by relational structures. *Discrete Applied Mathematics*, vol. 54, 1994, pp. 117–149.

[Don70] John DONER. – Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, n° 4, 1970, pp. 406–451.

[EH85] E. Allen EMERSON et Joseph Y. HALPERN. – Decision procedures and expressiveness in the Temporal Logic of Branching Time. *Journal of Computer and System Sciences*, vol. 30, n° 1, 1985, pp. 1–24.

[EH86] E. Allen EMERSON et Joseph Y. HALPERN. – “Sometimes” and “Not Never” revisited : on Branching versus Linear Time Temporal Logic. *Journal of the ACM*, vol. 33, n° 1, 1986, pp. 151–178.

[EL87] E. Allen EMERSON et Chin-Laung LEI. – Modalities for model checking : branching time strikes back. *Science of Computer Programming*, vol. 8, n° 3, 1987, pp. 275–306.

[ER91] Joost ENGELFRIET et Grzegorz ROZENBERG. – Graph grammars based on node rewriting : An introduction to NLC graph grammars. *In : Proc. 4th Int. Workshop on Graph Grammars, Bremen, 1990*, *Lecture Notes in Computer Science*, volume 532. pp. 12–23. – Springer-Verlag, Berlin/New York, 1991.

[FV59] Solomon FEFERMAN et Robert L. VAUGHT. – The First Order Properties Of Products Of Algebraic Systems. *Fund. Math.*, vol. 47, 1959, pp. 57–103.

[Gai82] Haim GAIFMAN. – On local and non-local properties. *In : Proc. of the Herbrand Symposium, Logic Colloqium '81*, éd. par J. STERN, pp. 105–135. – North Holland, Amsterdam, 1982.

[GPSS80] Dov M. GABBAY, Amir PNUELI, Saharon SHELAH et Jonathan STAVI. – On the temporal analysis of fairness. *In : Proc. 7th Ann. ACM Symp. on Principles of Programming Languages*, pp. 163–173. – 1980.

[GTW78] Joseph A. GOGUEN, James W. THATCHER et Eric G. WAGNER. – An Initial Approach to the Specification, Correctness and Implementation of Abstract Data Types. *In : Data Structuring*, éd. par R.T. YEH, *Current Trends in Programming Methodology*, volume 4, pp. 80–149. – Prentice Hall, 1978.

[HJJ+95] Jesper G. HENRIKSEN, Jakob JENSEN, Michael JØRGENSEN, Nils KLARLUND, Robert PAIGE, Theis RAUHE et Anders SANDHOLM. – MONA : Monadic second-order logic in practice. *In : Proc. 1st Inter. Workshop on Tools and Algorithms for Construction and Analysis of Systems, Aarhus, Denmark*, *Lecture Notes in Computer Science*, volume 1019, pp. 89–110. – mai 1995.

[HM85] Mathew HENNESSY et Robin MILNER. – Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM*, vol. 32, 1985, pp. 137–162.

[HS91] Joseph Y. HALPERN et Yoav SHOHAM. – A propositional Modal Logic of Time Intervals. *Journal of the ACM*, vol. 38, nº 4, 1991, pp. 935–962.

[HU79] John E. HOPCROFT et Jeffrey D. ULLMAN. – *Introduction to Automata Theory, Languages, and Computation.* – Addison-Wesley, 1979.

[Hun98] Hardi HUNGAR. – *Beyond finite-state model checking : Verifying large and infinite systems.* – Habilitation à diriger des recherches, Carl von Ossietzky Universität Oldenburg, 1998.

[Hun99] Hardi HUNGAR. – Model checking and higher-order recursion. *In : Proc. 24th Inter. Symp. on Mathematical Foundations of Computer Science, Szklarska Poreba, Poland*, éd. par M. KUTYLOWSKI, L. PACHOLSKI et T. WIERZBICKI, *Lecture Notes in Computer Science*, volume 1672. pp. 149–159. – Springer Verlag, septembre 1999.

[Kam68] H. W. KAMP. – *Tense logic and the theory of linear order.* – Thèse de Doctorat, Univ. of California, Los Angeles, CA, 1968.

[KC99] Teodor KNAPIK et Hugues CALBRIX. – Thue Specification and Their Monadic Second–Order Properties. *Fundamenta Informaticae*, vol. 39, 1999, pp. 305–325.

[Kla98] Nils KLARLUND. – Mona & fido : The logic-automaton connection in practice. *In : Proc. 11th Inter. Workshop on Computer Science Logic, Annual Conference of the EACSL, Aarhus, Denmark*, éd. par M. NIELSEN et W. THOMAS, *Lecture Notes in Computer Science*, volume 1414. pp. 311–326. – Springer Verlag, 1998.

[KM89] Robert. P. KURSHAN et Kenneth MCMILLAN. – A structural induction theorem for processes. *In : Proc. 8th Ann. ACM Symp. on Principles of Distributed Computing, Edmonton, Alberta, Canada*, pp. 239–247. – 1989.

[KMMG97] Peter Kelb, Tiziana Margaria, Michael Mendler et Claudia Gsottberger. – Mosel : A Flexible Toolset for Monadic Second-Order Logic. *In : Proc. 3rd Inter. Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Enschede, The Netherlands*, éd. par E. Brinksma, *Lecture Notes in Computer Science*, volume 1217, pp. 183–202. – mars 1997.

[Kna96] Teodor Knapik. – Spécifications de Thue. – 1996. IREMIA, Université de la Réunion. Résultats non-publiés.

[Kna97] Teodor Knapik. – *Domains of Word–functions and Thue Specifications.* – Rapport de recherche n° INF/96/11/05/a, IREMIA Université de La Réunion, 1997.

[KP98] Teodor Knapik et Étienne Payet. – The full quotient and its closure property for regular languages. *Information Processing Letters*, vol. 65, 1998, pp. 57–62.

[KP99] Teodor Knapik et Étienne Payet. – Synchronized Product of Linear Bounded Machines. *In : Proc. 12th Inter. Symp. on Fundamentals of Computation Theory, Iasi, Romania*, éd. par G. Ciobanu et G. Păun, *Lecture Notes in Computer Science*, volume 1684. pp. 362–373. – Springer Verlag, 1999.

[Kur87] Robert. P. Kurshan. – Reducibility in analysis of coordination. *Lecture Notes in Computer Science*, volume 103. pp. 19–39. – Springer Verlag, 1987.

[Lad77] Richard E. Ladner. – Application of model-theoretic games to discrete linear orders and finite automata. *Information and Control*, n° 33, 1977, pp. 281–303.

[LPY95] Kim G. Larsen, Paul Pettersson et Wang Yi. – Diagnostic model checking for real-time systems. *In : Proc. 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey*, *Lecture Notes in Computer Science*, volume 1066. pp. 575–586. – Springer Verlag, 1995.

[LPY97] Kim G. Larsen, Paul Pettersson et Wang Yi. – Uppaal in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, vol. 1, n° 1 et 2, 1997.

[LPZ85] Orna Lichtenstein, Amir Pnueli et Lenore D. Zuck. – The glory of the past. *In : Proc. Conf. on Logics of Programs*, *Lecture Notes in Computer Science*, volume 193. pp. 196–218. – Springer Verlag, Berlin, 1985.

[McM92] Kenneth L. McMillan. – *Symbolic Model Checking : An Approach to the State Explosion Problem.* – Thèse de Doctorat, Carnegie Mellon University, 1992. Publié par Kluwer sous le titre *Symbolic Model Checking*, 1993.

[McN66] Robert McNaughton. – Testing and generating infinite sequences by a finite automaton. *Information and Control*, n° 9, 1966, pp. 521–530.

[Mil80] Robin Milner. – *Calculus of Communicating Systems.* – Springer Verlag, 1980, *Lecture Notes in Computer Science*, volume 82.

[Mor00] Christophe Morvan. – On rational graphs. *In : Proc. FOSSACS'2000.* – 2000. À paraître dans Lecture Notes In Computer Science.

[Mos83] Ben MOSZKOWSKI. – *Reasoning about digital circuits.* – Thèse de Doctorat, Stanford Univ., 1983.

[MP83] Zohar MANNA et Amir PNUELI. – How to cook a proof system for your pet language. *In : Proc. 10th Ann. ACM Symp. on Principles of Programming Languages*, pp. 141–154. – 1983.

[MP89] Zohar MANNA et Amir PNUELI. – The anchored version of the temporal framework. *In : Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, éd. par J. W. de Bakker et AL., *Lecture Notes in Computer Science*, volume 345. pp. 201–284. – Springer Verlag, Berlin, 1989.

[MS81] David E. MULLER et Paul E. SCHUPP. – Pushdown automata, ends, second-order logic and reachability problems. *In : Proc. 13th Ann. ACM Symp. on the Theory of Computing*, pp. 46–54. – Milwaukee, 1981.

[MS83] David E. MULLER et Paul E. SCHUPP. – Groups, the theory of ends and context-free languages. *Journal of Computer and System Sciences*, vol. 26, n° 3, 1983, pp. 295–310.

[MS85] David E. MULLER et Paul E. SCHUPP. – The Theory of Ends, Pushdown Automata and Second–order Logic. *Theoretical Computer Science*, vol. 37, 1985, pp. 51–75.

[MS97] Alexandru MATEESCU et Arto SALOMAA. – Aspects of Classical Language Theory. *In : Word, Language, Grammar*, éd. par Grzegorz ROZENBERG et Arto SALOMAA, *Handbook of Formal Languages*, volume 1, pp. 175–251. – Springer Verlag, 1997.

[Niv79] Maurice NIVAT. – Sur la synchronisation des processus. *Revue technique Thomson-CSF*, vol. 11, 1979, pp. 899–919.

[Par70] David PARK. – Fixpoint induction and proof of program semantics. *Mach. Int.*, volume 5. pp. 59–78. – Edinburgh Univ. Press, 1970.

[Par74] David PARK. – *Finiteness is mu-ineffable.* – Theory of Computation Report n° 3, The University of Warwick, 1974.

[Pnu77] Amir PNUELI. – The Temporal Logic of programs. *In : 18th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 46–57. – 1977.

[Pnu84] Amir PNUELI. – In transition from global to modular reasoning about concurrent programs. *In : Logics and Models of Concurrent Systems*, éd. par K.R. APT. – Springer Verlag, Berlin, 1984.

[Pnu86] Amir PNUELI. – Applications of Temporal Logic to the specification and verification of reactive systems : a survey of current trends. *In : Current Trends in Concurrency : Overviews and Tutorials*, *Lecture Notes in Computer Science*, volume 224. – Springer Verlag, Berlin, 1986.

[Pra81] Vaughan R. PRATT. – A decidable μ-calculus. *In : Proc. 22nd Symp. on Foundations of Computer Science*, pp. 421–427. – 1981.

[QS82] Jean Pierre QUEILLE et Joseph SIFAKIS. – Specification and verification of concurrent programs in CESAR. *In : Proc. 5th Inter. Symp. on Programming*, *Lecture Notes in Computer Science*, volume 137. pp. 195–220. – Springer Verlag, Berlin, 1982.

[Rab69] Michael O. RABIN. – Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, n° 141, 1969, pp. 1–35.

[Rab77] Michael O. RABIN. – *Handbook of mathematical logic*, chap. Decidable Theories, pp. 595–629. – North-Holland Publishing Company, 1977. Edited by J. Barwise.

[Rau95] Antoine RAUZY. – Toupie = μ-calculus + constraints. *In : 7th Inter. Conf. on Computer Aided Verification, Liège, Belgium*, éd. par Pierre WOLPER, *Lecture Notes in Computer Science*, volume 939, pp. 114–126. – 1995.

[Roe74] Willem P. De ROEVER. – *Recursive program schemes : Semantics and proof theory.* – Thèse de Doctorat, Free University, Amsterdam, 1974.

[SB69] Dana SCOTT et J. W. De BAKKER. – A theory of programs. – 1969. Manuscript non publié, IBM, Vienne.

[SC85] A. Prasad SISTLA et Edmund M. CLARKE. – The complexity of Propositional Linear Temporal Logic. *Journal of the ACM*, vol. 32, n° 3, 1985, pp. 733–749.

[Sis85] A. Prasad SISTLA. – Characterizations of safety and liveness properties in Temporal Logic. *In : Proc. 4th Ann. ACM Symp. on Principles of Distributed Computing*, pp. 39–48. – 1985.

[SMSV83] Richard L. SCHWARTZ, P. Michael MELLIAR-SMITH et Friedrich VOGT. – An interval logic for higher-level temporal reasoning. *In : Proc. 2nd Ann. ACM Symp. on Principles of Distributed Computing*, pp. 173–186. – 1983.

[Tho79] Wolfgang THOMAS. – Star-free regular sets of ω-sequences. *Information and Control*, n° 42, 1979, pp. 148–156.

[Tho81] Wolfgang THOMAS. – A combinatorial approach to the theory of ω-automata. *Information and Control*, n° 48, 1981, pp. 261–283.

[Tho90] Wolfgang THOMAS. – Automata on infinite objects. *In : Formal Models and Semantics*, éd. par Jan van LEEUWEN, *Handbook of Theoretical Computer Science*, volume B, pp. 133–191. – Elsevier, 1990.

[Thu14] Axel THUE. – Probleme über veränderungen von Zeichenreihen nach gegebenen Regeln. *Skr. Vid. Kristiania, I Mat. Natuv. Klasse*, vol. 10, 1914, p. 34 pp.

[TW68] James W. THATCHER et Jesse B. WRIGHT. – Generalized finite automata with an application to a decision problem of second-order logic. *Math. Systems Theory*, n° 2, 1968, pp. 57–82.

[Wal96] Igor WALUKIEWICZ. – Pushdown processes : Games and model checking. *In : Proc. 8th Inter. Conf. on Computer Aided Verification, New Brunswick, NJ, USA*, *Lecture Notes in Computer Science*, volume 1102, pp. 62–74. – juillet 1996. To appear in Information and Computation.

[Wol83] Pierre Wolper. – Temporal logic can be more expressive. *Information and Control*, vol. 56, 1983, pp. 72–99.

[Wol86] Pierre Wolper. – Expressing interesting properties of programs in Propositional Temporal Logic. *In : Proc. 13th Ann. ACM Symp. on Principles of Programming Languages*, pp. 184–193. – 1986.