tel-00004890, version 1 - 19 Feb 2004

THÈSE

présentée par

HABERMEHL Peter

pour obtenir le grade de DOCTEUR
de l'UNIVERSITÉ JOSEPH FOURIER - GRENOBLE I
(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

(Spécialité : INFORMATIQUE)

SUR LA VÉRIFICATION DE SYSTÈMES INFINIS

Date de soutenance : 27 janvier 1998

Composition du jury :	Président	Jacques VOIRON
	Rapporteurs	Alain FINKEL
		Pierre WOLPER
	Examinateurs	André ARNOLD
		Ahmed BOUAJJANI
		Rachid ECHAHED
		Daniel KROB

Thèse préparée au sein du laboratoire VERIMAG.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

La conversation fut infinie entre les deux amies.

Stendhal

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Remerciements

Je tiens à remercier,

Monsieur Jacques Voiron, Professeur à l'Université Joseph Fourier (Grenoble I), de m'avoir fait l'honneur de présider le jury de cette thèse.

Messieurs Alain Finkel, Professeur à l'École normale supérieur de Cachan, et Pierre Wolper, Professeur à l'Université de Liège, pour avoir accepté de juger ce travail et d'en être rapporteurs.

Messieurs André Arnold, Professeur à l'Université de Bordeaux I, et Daniel Krob, Chargé de Recherche au CNRS de m'avoir fait l'honneur de participer au jury.

Messieurs Ahmed Bouajjani, Maître de Conférence à l'Université Joseph Fourier (Grenoble I), et Rachid Echahed, Chargé de Recherche au CNRS, qui ont dirigé cette thèse et sans lesquels ce travail n'aurait pu aboutir.

Cette thèse a été effectuée dans le laboratoire VERIMAG. Je tiens à remercier Monsieur Joseph Sifakis, Directeur de Recherche au CNRS, de m'avoir accueilli dans cette équipe.

Je tiens à remercier aussi tous ceux qui ont travaillé avec moi ces dernières années, en particulier Javier Esparza, ainsi que tous les membres de VERIMAG pour leur soutien amical. Je remercie en particulier les autres thésards.

tel-00004890, version 1 - 19 Feb 2004

Table des matières

1 Introduction **11**
1.1 Systèmes infinis 12
1.2 Méthodes de vérification algorithmiques 13
1.3 Propriétés non-régulières 15
1.4 Méthodes semi-algorithmiques 15
1.5 Organisation du document 17

2 Systèmes **19**
2.1 Préliminaires 19
2.1.1 Langages 19
2.1.2 L'arithmétique de Presburger 22
2.2 Systèmes de transitions étiquetées 24
2.3 Automates et ω-automates finis 25
2.3.1 ω-langages simples 27
2.4 Systèmes à pile 27
2.5 Les algèbres de processus 30
2.5.1 La syntaxe des algèbres PA, BPA, BPP, RP 30
2.5.2 Sémantique opérationnelle et langages générés 32
2.5.3 Forme normale 33
2.5.4 Notations et résultats préliminaires 35
2.6 Les réseaux de Petri 37
2.6.1 Les réseaux de Petri 37
2.6.2 Les systèmes d'addition de vecteurs avec états 39
2.6.3 Réseau BPP et SAVE 40
2.7 Semilinéarité 41
2.7.1 Définitions 41
2.7.2 Résultats 41
2.8 L'expressivité 46
2.8.1 Langages de mots finis 46
2.8.2 ω-langages 47

tel-00004890, version 1 - 19 Feb 2004

3 La complexité du μ-calcul linéaire pour les réseaux de Petri 49
3.1 Préliminaires . . . 50
3.1.1 Résultats de complexité sur les ω-automates . . . 50
3.1.2 Le μ-calcul propositionnel linéaire . . . 51
3.1.3 Les classes de complexité . . . 53
3.2 Résultat de complexité pour les SAVE . . . 54
3.2.1 Notation . . . 54
3.2.2 Problème de répétition d'un état de contrôle . . . 55
3.3 La borne supérieure . . . 59
3.4 La borne inférieure . . . 61

4 La logique temporelle linéaire contrainte 65
4.1 La logique CLTL . . . 66
4.1.1 Syntaxe . . . 67
4.1.2 Sémantique . . . 69
4.2 Expressivité . . . 70
4.2.1 Propriétés régulières . . . 71
4.2.2 Propriétés non-régulières . . . 73
4.3 Résultats d'indécidabilité pour CLTL . . . 75
4.4 Les fragments de CLTL . . . 77
4.4.1 Définitions . . . 77
4.4.2 Expressivité . . . 80
4.4.3 Les problèmes de satisfaisabilité et validité pour les fragments . . . 81
4.5 La décomposition de formules $\mathrm{CLTL}_{\Diamond}$ . . . 81
4.5.1 Forme normale . . . 81
4.5.2 Forme normale mono-contrainte . . . 83
4.5.3 Décomposition . . . 84
4.5.4 Satisfaisabilité et validité relative de $\mathrm{CLTL}_{\Diamond}$ et $\mathrm{CLTL}_{\Box}$ . . . 86
4.6 Le problème de la vérification pour $\mathrm{CLTL}_{\Box}$ . . . 88
4.6.1 Vérification des ω-automates à pile . . . 88
4.6.2 Vérification de processus PA . . . 88
4.6.3 Vérification de réseaux de Petri . . . 95
4.7 Conclusion . . . 96

5 Analyse d'automates communicants 99
5.1 Introduction . . . 99
5.2 Les automates communicants . . . 103
5.2.1 Définitions . . . 103
5.3 Diagrammes de décision de contenu de files contraints (CQDD) . . . 105
5.3.1 Les automates séquentiels . . . 106
5.3.2 Formule caractéristique . . . 111
5.3.3 Structures de représentation de contenus de files . . . 112
5.3.4 Expressivité . . . 113

tel-00004890, version 1 - 19 Feb 2004

5.3.5 Opérations de base et problèmes de décision 114
5.4 Représentation et manipulation d'ensembles de configurations 128
5.4.1 Représentation d'ensemble de configurations 128
5.4.2 Opérations de bases sur les ensembles de configurations 129
5.4.3 Calcul de l'effet d'un circuit 130
5.5 Analyse d'atteignabilité en avant et en arrière 134
5.6 Comparaison avec d'autres travaux existants 136

6 Conclusion 137

A Preuve du Théorème 5.4 139
A.1 Analyse de l'effet d'un circuit 139
A.1.1 Analyse d'un circuit avec taille croissante 139
A.1.2 Analyse d'un circuit avec taille décroissante 142
A.2 Fermeture par $post^*_\theta$ 145

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Chapitre 1

Introduction

Les systèmes qui interagissent avec leur environnement sont d'une importance capitale en informatique. Le comportement d'un tel système *réactif* est donné par une suite d'états et d'actions qui peut être infinie. La *spécification* décrit le comportement (infini) attendu du système. La *vérification* d'un système consiste à montrer qu'il satisfait sa spécification.

Les systèmes sont généralement infinis, dans le sens qu'ils ont un nombre infini d'états. Par exemple, le nombre d'états d'une machine de Turing est infini, puisque la taille de son ruban n'est pas bornée. Un des résultats fondamentaux sur les machines de Turing est l'indécidabilité du problème de l'arrêt. Le problème de l'arrêt est équivalent au problème d'atteignabilité, c'est-à-dire le problème de savoir si un état donné peut être atteint à partir de l'état initial. L'indécidabilité du problème de l'arrêt implique par conséquent que la plupart les problèmes "intéressants" de *vérification* pour ces systèmes sont indécidables.

Des systèmes qui n'ont pas la puissance des machines de Turing sont utilisés largement dans différents domaines de l'informatique. Pour ces systèmes, la vérification automatique (c.-à-d. algorithmique) de certaines propriétés est possible. La recherche concernant la vérification automatique de systèmes réactifs a commencé dans les années 70 avec l'étude des systèmes à nombre fini d'états. Il y a deux méthodes bien établies pour la vérification automatique des systèmes finis. D'une part la méthode basée sur la notion d'équivalence comportementale et d'autre part le *model-checking*.

Dans la première méthode, la spécification d'un système est donnée par un autre système fini, et la vérification consiste à montrer que les deux systèmes sont équivalents modulo une relation d'équivalence. Un exemple d'une telle relation est la bisimulation [Mil89, Par81].

Dans la méthode de *model-checking* la spécification est exprimée par des formules de *logiques temporelles*. Les logiques temporelles permettent de raisonner sur l'évolution d'un système dans le temps. Le problème de la vérification est de savoir si le système satisfait une propriété donnée par une formule de la logique. Pour les systèmes finis, beaucoup de résultats ont été prouvés pour différentes logiques temporelles et des outils performants ont été développés [CE81, QS82, CES83, VW86, McM93, Kur94]. Dans ce mémoire nous considérons la méthode du *model-checking*.

Les deux questions principales que nous nous posons dans cette thèse sont

tel-00004890, version 1 - 19 Feb 2004

> Pour quels systèmes infinis et quelles propriétés la vérification automatique est-elle possible ?
>
> Si le problème de la vérification est indécidable en général, est-ce qu'il existe des méthodes de semi-décision puissantes qui permettent dans certains cas d'analyser automatiquement un système ?

1.1 Systèmes infinis

Plusieurs formalismes couramment utilisés dans différents domaines de l'informatique définissent des systèmes infinis. Nous considérons en particulier les algèbres de processus et les automates d'états finis munis de structures de données non-bornées.

Considérons d'abord les algèbres de processus [BW90]. Un *processus* est décrit par un ensemble fini de *règles* sur un ensemble fini de *variables de processus*. Un processus décrit un système de transitions, où les états sont des *termes de l'algèbre* et les transitions sont étiquetées par des actions. Les opérations autorisées pour composer des termes dont les règles déterminent le pouvoir expressif de l'algèbre. L'algèbre de processus CCS [Mil89] par exemple a le pouvoir expressif des machines de Turing. Un petit nombre d'opérations très simples peut donner un formalisme qui permet de décrire des systèmes infinis. Il suffit par exemple de considérer le non-déterminisme ("+"), la composition séquentielle ("·") et la récursion. L'algèbre ainsi défini s'appelle BPA (*Basic Process Algebra*) [BW90]. Considérons par exemple un processus BPA qui modélise un compteur:

$$\begin{aligned} Z &= deb \cdot X \cdot Z \\ X &= inc \cdot Y \cdot X + fin \\ Y &= inc \cdot Y \cdot Y + dec \end{aligned}$$

Le système de transitions étiquetés généré par ce processus est donné figure 1.1. Intuitivement, le nombre de Y dans les termes indiquent la valeur du compteur.

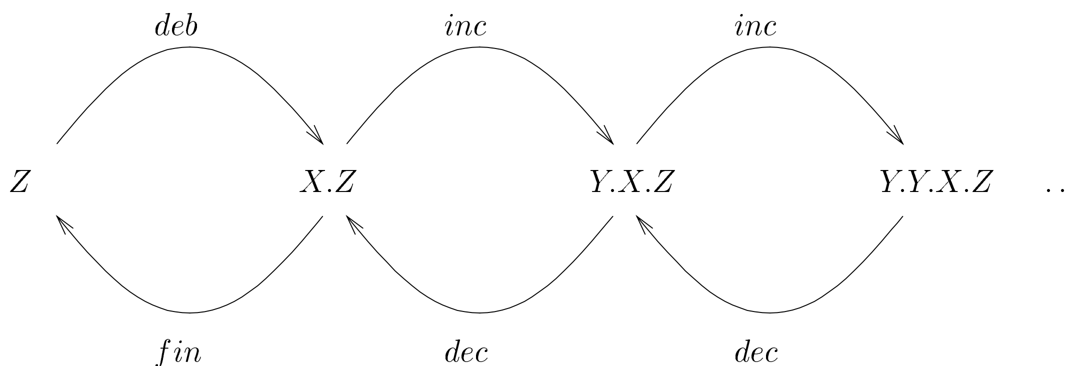


FIG. 1.1 – *Un processus BPA*

tel-00004890, version 1 - 19 Feb 2004

Si nous remplaçons dans BPA la composition séquentielle par la composition parallèle (asynchrone) et si nous ajoutons l'opération de préfixage ($a.t$), nous obtenons l'algèbre de processus BPP (Basic Parallel Processes) [Chr93]. Un processus BPP simple est par exemple donné par:

$$\begin{aligned} X &= a \cdot (X \| Y) + c \\ Y &= b \end{aligned}$$

Le système de transitions étiquetés généré par ce processus est donné figure 1.2.

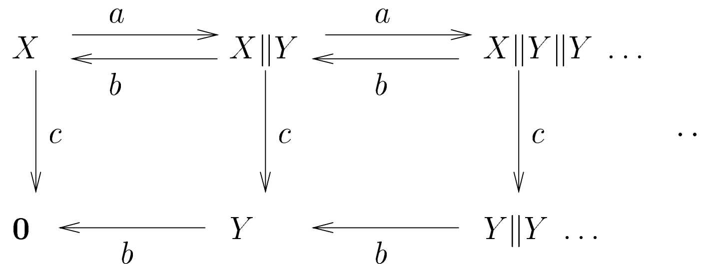


FIG. 1.2 – *Un processus BPP*

Nous obtenons l'algèbre PA (Process Algebra) [BK88] en autorisant le non-déterminisme, les compositions séquentielles et parallèles ainsi que la récursion. Il est possible de considérer des classes de processus encore plus générales: par exemple les *Process Rewrite Systems* [May98].

Un autre formalisme largement utilisé pour définir des systèmes infinis est donné par les automates finis munis de structures de données non-bornées. Les structures de données peuvent être séquentielles (piles ou files) ou des variables entières ou réelles. Une machine de Turing par exemple consiste en un automate fini avec un ruban non-borné. Les automates communicants (*Communicating Finite-State Machines* ou CFSM) introduits par Bochmann [Boc78] sont des automates finis qui communiquent d'une manière asynchrone via des files non-bornées. Ce modèle a la puissance d'une machine de Turing et est à la base des langages de spécification ESTELLE [ISO89] et SDL [CCI88]. Des exemples de systèmes qui n'ont pas la puissance d'une machine de Turing sont: les automates à pile, les réseaux de Petri [Pet62] (Système d'addition de vecteurs) qui peuvent être vus comme un automate fini avec des variables entières (avec des tests restreints sur les variables) et les automates temporisés [AD94] qui sont des automates qui manipulent des variables réelles.

1.2 Méthodes de vérification algorithmiques

Pour les systèmes infinis qui n'ont pas la puissance d'une machine de Turing une vérification algorithmique peut être envisagée. Ces dernières années un grand effort a été fait

pour étendre les résultats obtenus pour la vérification de systèmes finis vers les systèmes infinis.

Moller donne dans [Mol96] un résumé très détaillé des résultats obtenus jusqu'à présent pour la vérification automatique avec la méthode basée sur la notion d'équivalence comportementale.

Nous nous intéressons ici à la méthode de *model-checking*. La spécification d'un système est donnée par une formule de logique temporelle. On distingue deux familles de logiques temporelles: Les logiques *arborescentes* qui permettent de raisonner sur les arbres (infinis) et les logiques *linéaires* où on raisonne sur des ensembles de séquences (infinis). L'utilisation d'une logique temporelle comme langage de spécification a été introduite par Pnueli [Pnu77], qui a défini la logique LTL (*Linear temporal logic*). Plusieurs logiques temporelles linéaires et arborescentes ont été proposées ensuite. Dans le cadre linéaire la logique classique la plus expressive est le μ-calcul propositionnel linéaire [Koz83, Var88] (LTL en est une sous-classe) et dans le cadre arborescent le μ-calcul arborescent [Koz83].

Le μ-calcul linéaire a une caractérisation par les automates finis de Büchi, c'est-à-dire qu'étant donné une formule, l'ensemble des modèles est donné par le langage d'un ω-automate de Büchi. Cette relation forte entre logique et automate permet d'utiliser les résultats de la théorie des ω-automates pour la vérification de systèmes. Le problème de vérification est transformé vers un problème d'inclusion de ω-langage [Var96, VW86].

Le tableau 1.1 résume les résultats de décidabilité et de complexité obtenus jusqu'à présent pour le problème de la vérification de différentes classes de systèmes infinis par rapport aux μ-calculs.

$\models$	μ-calcul arborescent	μ-calcul linéaire,LTL
PA	indécidable[Esp97]	indécidable [Hab98]
automates à pile	DEXPTIME-complet[Wal96]	DEXPTIME-complet [BEM97]
BPA	DEXPTIME [BS97]	DEXPTIME-complet [May98]
réseaux de Petri et BPP	indécidable [Esp97]	EXPSPACE-complet (système)[Hab98] PSPACE-complet (formule)[Hab98]

TAB. 1.1 – *Le problème de la vérification*

Dans ce mémoire nous montrons notre contribution à ce tableau. Nous montrons l'indécidabilité du problème de model-checking de LTL (et par conséquent du μ-calcul linéaire) par rapport à PA (voir section 4.6.2). Nous analysons aussi la complexité de la vérification du μ-calcul linéaire pour les réseaux de Petri et les BPP (voir chapitre 3). Nous montrons que ce problème est EXPSPACE-complet dans la taille du système et PSPACE-complet dans la taille de la formule.

Une étude très complète de la décidabilité et de la complexité du problème de vérification de systèmes infinis par rapport à plusieurs logiques temporelles linéaires et arborescentes a récemment été faite par Mayr [May98]. Deux autres résumés peuvent être trouvés dans [Esp96, BE97].

tel-00004890, version 1 - 19 Feb 2004

1.3 Propriétés non-régulières

Tous les travaux existants mentionnés ci-dessus considèrent le problème de la vérification des systèmes infinis par rapport à des propriétés *régulières*, c'est-à-dire les propriétés définies par des automates *finis* sur les séquences ou des arbres infinis [Tho90]. Cependant, il existe des propriétés significatives de systèmes infinis qui ne sont pas régulières. Par exemple, deux propriétés importantes du processus de la figure 1.1 sont données par:

1. *Entre deb et fin il y a le même nombre de inc que de dec.*
2. *Le nombre de dec n'est jamais plus grand que le nombre de inc.*

Ces exemples montrent que des propriétés significatives de systèmes infinis sont essentiellement des propriétés temporelles imposant des contraintes sur le nombre d'occurrences d'événements. Il est évident que ces propriétés ne peuvent pas être exprimées par les logiques de spécification régulières (logiques temporelles et μ-calculs propositionnels). Nous introduisons dans le chapitre 4 la logique temporelle linéaire CLTL (*Constraint Linear Temporal Logic*) qui permet d'exprimer des contraintes linéaires sur le nombre d'occurrences d'événements. Le pouvoir expressif "régulier" de CLTL est maximal dans le sens qu'elle peut exprimer toutes les propriétés ω-régulières sur les séquences. CLTL permet en plus de définir des propriétés non-régulières grâce à l'utilisation de contraintes arithmétiques exprimées en arithmétique de Presburger [Pre29].

Par exemple, la propriété 1 ci-dessus peut être exprimée dans CLTL avec la formule

$$\Box\,(\mathrm{deb} \Rightarrow\ [x, y : \mathrm{inc}, \mathrm{dec}].\ \Box\,(\mathrm{fin} \Rightarrow (x = y)))$$

Intuitivement, la construction $[x, y : \mathrm{inc}, \mathrm{dec}]$ permet d'initialiser deux compteurs qui compte le nombre d'occurrences de *inc* respectivement *dec*. La formule $x = y$ permet de tester si le nombre d'occurrences de *inc* et égal au nombre d'occurrences de *dec* à partir du point où les compteurs ont été initialisés. De la même manière nous pouvons exprimer la propriété 2 avec

$$\Box\,(\mathrm{deb} \Rightarrow\ [x, y : \mathrm{inc}, \mathrm{dec}].\ (x \geq y)\,\mathcal{U}\,\mathrm{fin})$$

Nous étudions le problème de la vérification de plusieurs classes de systèmes infinis par rapport à la logique CLTL et à plusieurs de ses fragments. Nous obtenons plusieurs résultats de décidabilité du problème de la vérification pour des classes de systèmes infinis (réseaux de Petri, automates à pile) et pour des fragments de CLTL qui sont plus expressifs que les logiques de spécifications classiques.

Nous abordons ensuite le problème de la vérification des automates communicants.

1.4 Méthodes semi-algorithmiques

Les automates communicants (CFSM) ont le même pouvoir expressif que les machines de Turing [BZ83]. La vérification automatique est donc en général indécidable. Nous montrons dans ce mémoire comment analyser ces systèmes d'une manière semi-algorithmique.

tel-00004890, version 1 - 19 Feb 2004

C'est-à-dire nous obtenons des algorithmes qui, s'ils s'arrêtent, permettent d'analyser certains aspects du système. Nous appliquons le principe de l'analyse symbolique.

Le calcul des états atteignables est souvent à la base de l'analyse d'un système. Si on veut vérifier que tous ses comportements sont *sûrs*, il suffit de tester que l'intersection de l'espace des états atteignables à partir des états initiaux avec l'ensemble des états *mauvais* est vide. Ce problème peut aussi être résolu en testant que l'intersection des états initiaux avec tous les prédécesseurs des mauvais états est vide. Le calcul des prédécesseurs ou des successeurs d'un ensemble d'états S donné est donc à la base de beaucoup de techniques de vérification. Pour simplifier la présentation nous ne considérons dans la suite que le calcul des successeurs (le calcul des prédécesseurs est symétrique).

L'ensemble de tous les successeurs d'un ensemble d'états S peut être vu comme la limite de la séquence *infinie* croissante $(X_i)_{i \geq 0}$ avec $X_{i+1} = X_i \cup post(X_i)$, $X_i = S$ et $post(X_i)$ est l'ensemble des successeurs immédiats de X_i. La limite de la séquence est atteinte si $X_i = X_{i+1}$.

Dans les systèmes finis on atteint toujours la limite puisque tous les ensembles sont finis. Par contre pour les systèmes infinis, les X_i sont en général infinis et la séquence $(X_i)_{i \geq 0}$ ne converge pas en un nombre fini de pas. Pour vérifier les systèmes infinis nous avons donc besoin de structures finis qui permettent de représenter un nombre infini d'états. Pour pouvoir calculer la séquence $(X_i)_{i \geq 0}$ et tester $X_i = X_{i+1}$, ces structures doivent être effectivement fermées par union, intersection et la fonction *post* et leur problèmes d'inclusions et du vide doivent être décidables. Un autre problème est la convergence de la séquence $(X_i)_{i \geq 0}$. Pour faire face à ce problème nous considérons *l'accélération exacte* du calcul de la limite. Cette technique définit une autre séquence qui a la même limite que $(X_i)_{i \geq 0}$ mais dont le calcul termine dans des cas où $(X_i)_{i \geq 0}$ ne termine pas.

Pour l'analyse des CFSM Boigelot et Godefroid [BG96] (voir aussi [BGWW97]) proposent des structures, appelées *Diagrammes de décision de contenu de file* (QDD). Un ensemble de contenus de plusieurs files est représenté par un automate d'états finis. Ils adaptent une technique d'accélération introduite dans [BW94] pour des systèmes avec variables entières. Elle est basée sur la notion de *méta-transition*. Une méta-transition correspond a un circuit θ dans le graphe de contrôle du système et le calcul de la limite est accéléré en ajoutant à chaque pas tous les successeurs après un nombre quelconque de répétitions du circuit. Cette opération est appelé $post_\theta^*$.

Le problème est de déterminer pour quels circuits θ la classe des structures de représentation est fermée par $post_\theta^*$. Cet ensemble de circuits est caractérisé dans [BGWW97] pour les QDD. Il est assez restreint pour pouvoir garantir que l'image d'un ensemble régulier par $post_\theta^*$ est aussi un ensemble régulier. Par exemple un circuit qui envoie le même message à deux files n'est pas dans cet ensemble.

Dans ce mémoire nous proposons dans le chapitre 5 une généralisation de l'approche de [BG96] et [BGWW97] en considérant l'accélération exacte pour *chaque* circuit dans le graphe de transitions du système. Le problème principal de cette nouvelle approche vient du fait que l'ensemble des états atteignables par un circuit général peut être non-régulier. Par conséquent, nous avons besoin de structures de représentation qui sont plus expressives que les automates finis.

tel-00004890, version 1 - 19 Feb 2004

Nous proposons une structure, appelée diagrammes de décision de contenu de files contraints (CQDD). Cette structure combine une sous-classe des automates finis avec des contraintes linéaires sur le nombre de fois que les transitions dans leurs calculs accepteurs sont prises. Nous montrons que les CQDD's satisfont toutes les propriétés d'une "bonne" structure de représentation (fermeture par union, intersection, etc.) Le résultat principal est que les CQDD's sont fermés par $post_\theta^*$ pour chaque circuit θ. Ce résultat permet de définir un semi-algorithme général d'analyse d'atteignabilité paramétré par l'ensemble de circuits considérés pour l'accélération.

1.5 Organisation du document

Dans le chapitre 2 nous donnons quelques définitions de base et introduisons les systèmes finis et infinis que nous considérons dans ce mémoire. Dans le chapitre 3 nous étudions la complexité du problème de *model-checking* des réseaux de Petri par rapport au μ-calcul propositionelle linéaire. Le chapitre 4 introduit plusieurs logiques temporelles linéaires avec contraintes qui permettent de spécifier des propriétés non-regulières. Nous montrons des résultats de décidabilité du problème de la vérification de ces logiques pour plusieurs classes de systèmes infinis. Ensuite nous analysons dans le chapitre 5 le problème de la vérification pour les automates communicants avant de conclure dans le chapitre 6.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Chapitre 2

Systèmes

Dans ce chapitre nous présentons les systèmes que nous considérons dans ce mémoire, et nous établissons quelques résultats préliminaires les concernant. Nous commençons par donner les notations et définitions de base sur les langages et l'arithmétique de Presburger. Ensuite nous introduisons les systèmes de transitions étiquetées. Nous introduisons les automates et ω-automates finis et les automates et ω-automates à pile. Ensuite, nous définissons les algèbres de processus que nous considérons et nous donnons des résultats préliminaires les concernant. Ensuite, nous introduisons les réseaux de Petri et pour conclure ce chapitre nous définissons les systèmes semilinéaires et nous analysons l'expressivité de toutes les classes de systèmes introduits dans ce chapitre.

2.1 Préliminaires

Dans cette section nous donnons les définitions de base sur les langages et l'arithmétique de Presburger.

2.1.1 Langages

Nous donnons quelques définitions de base sur les langages.

Définition 2.1 (séquences, langages) :

> Soit Σ un alphabet fini. Alors Σ^* est l'ensemble des séquences finies sur Σ. Σ^ω est l'ensemble des séquences infinies sur Σ. Un langage sur Σ est un sous-ensemble de Σ^* et un ω-langage L est un sous-ensemble de Σ^ω. Nous écrivons Σ^∞ pour $\Sigma^* \cup \Sigma^\omega$.

Définition 2.2 (Concaténation) :

> Étant donnés deux langages $L_1, L_2 \in \Sigma^*$, la concaténation de L_1 et L_2 est $L_1.L_2 := \{w_1.w_2 \ : \ w_1 \in L_1 \text{ et } w_2 \in L_2\}$

Définition 2.3 (Dérivation à gauche) :

Étant donnés deux langages $L_1, L_2 \in \Sigma^*$, la dérivée à gauche de L_1 par L_2 est $L_2^{-1}.L_1 := \{w \in \Sigma^* \ : \ \exists w' \in L_2.\ w'.w \in L_1\}$.

Définition 2.4 (Expressions régulières) :

Étant donné un alphabet fini Σ, les *expressions régulières* sont les expressions formées par les règles suivants:

- $\emptyset$, ϵ et chaque $a \in \Sigma$ sont des expressions régulières
- si α et β sont des expressions régulières, alors $\alpha + \beta$, $\alpha\beta$ et α^* sont aussi des expressions régulières.

Le langage défini par une expression régulière est défini comme habituellement.

Définition 2.5 (Langage sans hauteur d'étoile) :

Un langage $L \in \Sigma^*$ est *sans hauteur d'étoile* s'il peut être généré par les opérations booléennes et la concaténation à partir d'ensembles finis.

Une séquence infinie $\sigma \in \Sigma^\omega$ peut être vue comme une fonction de $I\!N$ vers Σ. σ est donc égal à $\sigma(0)\sigma(1)\cdots$. Une séquence fini $\sigma \in \Sigma^\omega$ peut être vue de la même manière comme une fonction partielle de $I\!N$ vers Σ.

Définition 2.6 $(\sigma(i,j), Pref(\sigma))$:

Soit $\sigma \in \Sigma^\infty$. Étant donnés i et j avec $i, j \in I\!N$ et $i \leq j \leq |\sigma|$, nous écrivons $\sigma(i,j)$ pour la séquence finie $\sigma(i)\cdots\sigma(j)$ (avec $\sigma(i,i) = \sigma(i)$). Étant donnée une séquence $\sigma \in \Sigma^\omega$, nous écrivons $Pref(\sigma)$ pour l'ensemble des séquences finies qui sont des préfixes finis de σ, i.e.,

$$Pref(\sigma) = \{\sigma(0,i) \ : \ 0 \leq i < |\sigma| + 1\}$$

Nous avons besoin des notions et Lemmes suivants pour les résultats du chapitre 4. Pour les trois définitions suivantes, soient $\mathcal{P}$ un ensemble fini, $\Sigma = 2^{\mathcal{P}}$, $\mathcal{P}'$ un ensemble fini avec $\mathcal{P}' \supseteq \mathcal{P}$ et $\Sigma' = 2^{\mathcal{P}'}$.

Définition 2.7 (Projection) :

Étant donnée une séquence $\sigma' \in (\Sigma')^\infty$, la *projection* de σ sur Σ, représentée par $\sigma'|_\Sigma$, est la séquence $\sigma \in \Sigma^\infty$ telle que pour chaque $i \geq 0$, $\sigma(i) = \sigma'(i) \cap \mathcal{P}$.

Définition 2.8 (Cylindrification) :

Étant donnée une séquence $\sigma \in (\Sigma)^\infty$, la *cylindrification* de σ vers Σ', représentée par $\widetilde{\sigma}$ est l'ensemble de séquences $\sigma' \in (\Sigma')^\infty$ tel que $\sigma = \sigma'|_\Sigma$.

tel-00004890, version 1 - 19 Feb 2004

Les deux définitions précédentes sont généralisées à des ensemble de séquences comme suit:

Définition 2.9 :

Soit $L' \subseteq (\Sigma')^\infty$. La projection de L' sur Σ est définie par:

$$L'|_\Sigma = \{\sigma'|_\Sigma \ : \ \sigma' \in L'\}$$

Soit $L \subseteq (\Sigma')^\infty$. La cylindrification de L vers Σ' est définie par:

$$\widetilde{L} = \bigcup_{\sigma \in L} \widetilde{\sigma}$$

Nous avons les résultats suivants pour la projection. Les preuves des trois lemmes suivants sont triviales.

Lemme 2.1 :

Soit $L \subseteq (\Sigma')^\omega$ tel que pour tout $\sigma \in L$ nous avons $\forall i \in I\!N.\ \sigma(i) \cap \Sigma \neq \emptyset$. Alors $L = \emptyset$ si et seulement si $L|_\Sigma = \emptyset$.

Lemme 2.2 :

Soit $L_1, L_2 \subseteq (\Sigma')^\omega$. Alors $(L_1 \cup L_2)|_\Sigma = L_1|_\Sigma \cup L_2|_\Sigma$

Lemme 2.3 :

Soit $L_1, L_2 \subseteq (\Sigma')^\omega$. Alors $(L_1 \cap L_2)|_\Sigma \subseteq L_1|_\Sigma \cap L_2|_\Sigma$. L'inverse n'est généralement pas vrai.

Lemme 2.4 :

Soient $L_1, L_2 \subseteq \Sigma^\omega$ et $L'_2 \subseteq (\Sigma')^\omega$ tel que $L_2 = L'_2|_\Sigma$. Alors, $L_1 \cap L_2 = (\widetilde{L_1} \cap L'_2)|_\Sigma$.

Preuve:

- Soit $\sigma \in L_1 \cap L_2$. Alors, puisque $L_2 = L'_2|_\Sigma$, $\sigma \in L_2$ implique $\exists \sigma' \in L'_2$ avec $\sigma'|_\Sigma = \sigma$. En plus $\sigma' \in \widetilde{L_1}$ par définition. Donc $\sigma' \in \widetilde{L_1} \cup L_2$ et $\sigma'|_\Sigma = \sigma$. Il s'en suit que $\sigma \in (\widetilde{L_1} \cap L'_2)|_\Sigma$.
- Soit $\sigma \in (\widetilde{L_1} \cap L'_2)|_\Sigma$. En utilisant le Lemme 2.3 il s'en suit que $\sigma \in \widetilde{L_1}|_\Sigma$ et $\sigma \in L'_2|_\Sigma$. Alors par définition $\sigma \in L_1$, et parce que $L_2 = L'_2|_\Sigma$, nous avons aussi $\sigma \in L_2$.

□

tel-00004890, version 1 - 19 Feb 2004

2.1.2 L'arithmétique de Presburger

Dans cette section nous donnons la définition de l'arithmétique de Presburger [Pre29] et les ensembles semilinéaires.

Soit $\mathcal{V}$ un ensemble de variables. Nous utilisons $x, y, \ldots$ pour des éléments de $\mathcal{V}$.

Définition 2.10 (Terme de l'arithmétique de Presburger) :

Les *termes de l'arithmétique de Presburger* sont donnés par

$$t ::= 0 \mid 1 \mid x \mid t \Leftrightarrow t \mid t + t$$

Nous utilisons des abréviations pour les constantes $k \in \mathbb{N}$ et pour la multiplication avec constante kt.

Définition 2.11 (Formule de Presburger) :

L'ensemble des *formules de Presburger* est définit par

$$f ::= t \leq t \mid \neg f \mid f \vee f \mid \exists x.\ f$$

Notation 2.1 :

Nous utilisons les abréviations usuelles suivantes:

$$\begin{aligned} f_1 \wedge f_2 &= \neg(f_1 \vee f_2) \\ f_1 \Rightarrow f_2 &= \neg f_1 \vee f_2 \\ f_1 \Leftrightarrow f_2 &= (f_1 \Rightarrow f_2) \wedge (f_2 \Rightarrow f_1) \\ \forall x.\ f &= \neg \exists x.\ \neg f \end{aligned}$$

La notion de variable libre est définie comme d'habitude. Nous écrivons $f(x_1, \ldots, x_n)$ pour indiquer que f contient $x_1, \ldots, x_n$ comme variables libres. La sémantique des formules de Presburger est définie de manière standard.

Définition 2.12 (Valuation) :

Étant donné un ensemble de variables $V = \{x_1, \ldots, x_n\} \subseteq \mathcal{V}$, une *valuation* E est une fonction $E : V \rightarrow \mathbb{N}$ (aussi représentée par un vecteur $(k_1, \ldots, k_n) \in \mathbb{N}^n$) qui associe à chaque variable x_i un entier k_i.

tel-00004890, version 1 - 19 Feb 2004

Définition 2.13 (Relation de satisfaction) :

Soit $f(x_1, \ldots, x_n)$ une formule de Presburger et E une valuation. E *satisfait* f, représenté par $E \vdash f$, si $f(E)$ est vraie.

Définition 2.14 (ensemble caractéristique) :

Soit f une formule de Presburger avec n variables libres. $\langle\!| f |\!\rangle \subseteq I\!N^n$ est l'ensemble des valuations qui satisfont f.

Définition 2.15 (Formule valide, satisfaisable) :

Une formule de Presburger f avec n variables libres est *satisfaisable* (resp. *valide*), ssi $\langle\!| f |\!\rangle \neq \emptyset$ (resp. $\langle\!| f |\!\rangle = I\!N^n$).

Théorème 2.1 (Presburger[Pre29]) :

Les problèmes de satisfaisabilité et de validité d'une formule de Presburger sont décidables.

L'ensemble $\langle\!| f |\!\rangle$ des valuations qui satisfont une formule de Presburger f peut être caractérisé par un ensemble semilinéaire, que nous définissons ci-dessous.

Définition 2.16 (Ensemble linéaire) :

Un *ensemble linéaire* est un sous-ensemble de $I\!N^n$ de la forme

$$\{\vec{v} + k_1\vec{u}_1 + \cdots + k_m\vec{u}_m \ : \ k_1, \ldots, k_m \in I\!N\}$$

où $n, m > 0$ et $\vec{v}, \vec{u}_1, \cdots, \vec{u}_m \in I\!N^n$.

Définition 2.17 (Ensemble semilinéaire) :

Un *ensemble semilinéaire* est une union finie d'ensembles linéaires.

Nous avons la relation suivante entre les formules de Presburger et les ensembles semilinéaire (pour une preuve voir par exemple [Har78]).

Lemme 2.5 ([Har78]) :

Pour chaque formule de Presburger f, l'ensemble caractéristique $\langle\!| f |\!\rangle$ est semilinéaire. Et chaque sous-ensemble semilinéaire $\mathcal{L}$ de $I\!N^n$ peut être caractérisé par une formule de Presburger g avec n variables libres, $\langle\!| g |\!\rangle = \mathcal{L}$.

Corollaire 2.1 :

Les ensembles semilinéaires sont fermés par intersection, union et négation.

tel-00004890, version 1 - 19 Feb 2004

2.2 Systèmes de transitions étiquetées

Chaque système que nous considérons dans ce mémoire peut être décrit comme un système de transitions étiquetées.

Définition 2.18 (Système de transitions étiquetées) :

Un *système de transitions étiquetées* est un quadruplet $\mathcal{S} = (Q, \Sigma, q_0, \delta)$ où:

- Q est un ensemble d'états,
- Σ est un ensemble fini d'étiquettes, l'*alphabet*
- q_0 est l'état initial,
- et $\delta \subseteq Q \times \Sigma \times Q$ est la relation de transition.

Pour $(q_1, a, q_2) \in \delta$ nous écrivons aussi $q_1 \xrightarrow{a}_\delta q_2$ ou $q_1 \xrightarrow{a} q_2$, si δ est clairement donné par le contexte.

Définition 2.19 (Système de transitions étiquetées déterministe) :

Un système de transitions étiquetées est *déterministe* si
$\forall q \in Q\ \forall a \in \Sigma.$ il existe au plus un $q' \in Q$ avec $(q, a, q') \in \delta$.

Définition 2.20 (Système de transitions étiquetées fini) :

Un système de transitions étiquetées (Q, Σ, q_0, δ) est *fini* si l'ensemble Q est fini.

Définition 2.21 (calcul) :

Étant donnés un système de transitions étiquetées $\mathcal{S} = (Q, \Sigma, q_0, \delta)$ et une séquence $\sigma \in \Sigma^\infty$, un *calcul* de $\mathcal{S}$ sur σ est une séquence $\rho \in Q^\infty$ avec

- $\rho(0) = q_0$
- $\forall i \geq 0.\ (\rho(i), \sigma(i), \rho(i+1)) \in \delta$ si $|\sigma| = \infty$, sinon
- $\forall i \in \{0, \ldots, |\sigma| \Leftrightarrow 1\}.\ (\rho(i), \sigma(i), \rho(i+1)) \in \delta$.

À chaque calcul $\rho \in Q^\omega$ est associé la séquence $\rho' \in \delta^\omega$ des transitions correspondantes. Un calcul ρ sur un $\sigma \in \Sigma^*$ est *terminant* si pour tout $q \in Q$ et $a \in \Sigma$ il n'existe pas de transition $(\rho(|\sigma|), a, q) \in \delta$.

tel-00004890, version 1 - 19 Feb 2004

2.3 Automates et ω-automates finis

Dans cette section, nous définissons les automates finis et les ω-automates finis ainsi que les langages associés à ces automates.

Définition 2.22 (Automate fini) :

Un *automate fini* sur Σ est un quintuplet $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, où

- (Q, Σ, q_0, δ) est un système de transitions étiquetées fini, noté par $\mathcal{S}_\mathcal{A}$
- $F \subseteq Q$ est l'ensemble d'*états terminaux*

Définition 2.23 (Langage d'un automate fini) :

Étant donné un automate fini $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, le langage de $\mathcal{A}$, noté $L(\mathcal{A})$, est donné par l'ensemble de séquences σ telles qu'il existe un calcul ρ de $\mathcal{S}_\mathcal{A}$ sur σ tel que $\rho(|\sigma|) \in F$. Ce calcul est appelé *calcul accepteur*.

Définition 2.24 (Circuit, Boucle) :

Soit $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ un automate fini. Un *circuit* de taille $n \geq 1$ est une suite d'états $q_1, \ldots, q_n$ de Q tel que $q_1 = q_n$ et $\forall i \in \{1, \ldots, n\}\ \exists a \in \Sigma\ q_i \xrightarrow{a}_\delta q_{i+1}$. Une *boucle* est un circuit de taille 1.

Définition 2.25 (Automate fini déterministe, complet) :

Un automate fini $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ est *déterministe* si $\mathcal{S}_\mathcal{A}$ est déterministe. $\mathcal{A}$ est *complet* ssi $\forall q \in Q\ \forall a \in \Sigma\ \exists q' \in Q\ (q, a, q') \in \delta$.

Nous avons le lemme bien connu suivant:

Lemme 2.6 :

Pour chaque automate fini $\mathcal{A}$, il existe un automate fini $\mathcal{A}'$ complet avec $L(\mathcal{A}) = L(\mathcal{A}')$.

Définition 2.26 (ω-automate) :

Un ω-automate d'états finis est un quintuplet $\mathcal{A} = (Q, \Sigma, q_0, \delta, C)$ où

- (Q, Σ, q_0, δ) est un système de transitions étiquetées fini, représenté par $\mathcal{S}_\mathcal{A}$ et
- C est une condition d'acceptation (voir tableau 2.1).

Définition 2.27 (ω-automate déterministe) :

Un ω-automate $\mathcal{A} = (Q, \Sigma, q_0, \delta, C)$ est *déterministe* si $\mathcal{S}_\mathcal{A}$ est déterministe.

tel-00004890, version 1 - 19 Feb 2004

Type	Syntaxe	Sémantique
Büchi	$F \subseteq Q$	$Inf(\rho) \cap F \neq \emptyset$
Rabin	$\bigvee_i L_i \wedge \neg U_i$	$\exists i : Inf(\rho) \cap L_i \neq \emptyset \wedge Inf(\rho) \cap U_i = \emptyset$
Streett	$\bigwedge_i L_i \rightarrow U_i$	$\forall i : Inf(\rho) \cap L_i = \emptyset \vee Inf(\rho) \cap U_i \neq \emptyset$

où L_i et U_i (le couple accepteur) sont des sous-ensembles de Q.

Tab. 2.1 – *Les conditions d'acceptation*

Définition 2.28 (calcul accepteur) :

Soit $\mathcal{A} = (Q, \Sigma, q_0, \delta, C)$ un ω-automate. Soit $\rho \in Q^\omega$ un calcul de $\mathcal{S}_\mathcal{A}$. Nous écrivons $Inf(\rho)$ pour l'ensemble d'états tel que $\exists^\infty i \in I\!N$ avec $\rho(i) = q$. Un chemin ρ est *accepteur* s'il satisfait une des conditions d'acceptation données dans le tableau 2.1.

Nous appelons ω-automate de Büchi (resp. Rabin, Streett) un ω-automate avec une condition d'acceptation de Büchi (resp. Rabin, Streett).

Définition 2.29 (ω-langage d'un ω-automate) :

Le *ω-langage* d'un ω-automates $\mathcal{A}$, noté $L(\mathcal{A})$, est l'ensemble des séquences $\sigma \in \Sigma^\omega$ tel que $\mathcal{S}_\mathcal{A}$ a un calcul accepteur sur σ.

Nous avons le théorème suivant pour les ω-automates non-déterministes.

Théorème 2.2 ([Tho90]) :

Les automates (non-déterministes) de Büchi, Rabin et Streett reconnaissent la même classe de langages.

Définition 2.30 (ω-langage régulier) :

Un ω-langage $L \in \Sigma^\omega$ est appelé *ω-régulier* si il existe un ω-automate de Büchi $\mathcal{A}$ avec $L(\mathcal{A}) = L$.

Définition 2.31 (ω-langage sans hauteur d'étoile) :

Un ω-langage est *sans hauteur d'étoile* si L est une union finie d'ensembles de la forme $L_1 L_2^\omega$, où $L_1, L_2 \in \Sigma^*$ sont sans hauteur d'étoile, et $L_2 L_2 \subseteq L_2$.

Le lemme suivant est de [Tho90]:

Lemme 2.7 :

La classe des langages ω-réguliers (resp. ω-langages sans hauteur d'étoile) est fermée par toutes les opérations booléennes.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

2.3.1 ω-langages simples

Nous introduisons une classe de ω-langages qui est moins expressive que les ω-langages sans hauteur d'étoile. Nous appelons cette classe ω-régulier simple.

Définition 2.32 (ω-langage régulier simple) :

Un ω-langage $L \subseteq \Sigma^\omega$ est appelé *ω-régulier simple* s'il existe un automate de Büchi $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ tel que tous les circuits de $\mathcal{A}$ sont des boucles et $L(\mathcal{A}) = L$.

Lemme 2.8 :

La classe des langages ω-réguliers simples est fermée par intersection et union mais pas par complémentation.

Preuve:

- La fermeture par union est triviale (Les automates sont non-déterministes).
- La fermeture par intersection: Soit $\mathcal{A}_1$ (avec condition d'acceptation F_1) et $\mathcal{A}_2$ (avec condition d'acceptation F_2) deux automates de Büchi tels que chaque circuit dans ces automates est une boucle. En utilisant la construction standard (voir par exemple [Var96]) pour construire l'intersection de deux ω-automates de Büchi, l'automate construit peut contenir un circuit qui n'est pas une boucle. Mais nous pouvons utiliser la construction standard pour l'intersection des *automates finis*. La condition d'acceptation de l'automate produit est donné par $F_1 \times F_2$. La propriété essentielle pour montrer que l'automate produit accepte exactement $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ est: si un calcul de $\mathcal{A}_1$ ou $\mathcal{A}_2$ quitte un état accepteur, il n'y peut jamais revenir (tous les circuits sont des boucles). Cela n'est pas vrai en général pour tous les ω-automates de Büchi.
- Non-fermeture par complémentation: Soit $\Sigma = \{a, b\}$, et $L = \Sigma^*.b^\omega$. Alors $\Sigma^* \setminus L$ est l'ensemble de séquences avec un nombre infini de a. Pour décrire ce langage nous avons besoin d'un automate avec un circuit avec deux états.

□

2.4 Systèmes à pile

Dans cette section nous définissons les automates à pile, les ω-automates à pile et les grammaires hors-contextes ainsi que les langages définis par eux.

Définition 2.33 (Automate à pile (PDA)) :

Un *automate à pile* est un sextuplet $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, où

- Q est un ensemble fini d'états,

– Σ est un alphabet fini,
– , est un alphabet de pile
– $\delta \subset ((Q \times \Sigma^* \times ,) \times (Q \times , ^*))$ est la relation de transition,
– q_0 est l'état initial,
– Z_0 est le symbole initiale de la pile,

Une *configuration* d'un automate à pile est un couple (q, γ), où $q \in Q$ et $\gamma \in , ^*$.

Définition 2.34 :

Soit $\mathcal{A} = (Q, \Sigma, , , \delta, q_0, Z_0)$ un automate à pile. Le système de transition étiquetées $\mathcal{S}_\mathcal{A} = (Q', \Sigma', q_0', \delta')$ associé à $\mathcal{A}$, est donné par

– $Q' = Q \times , ^*$,
– $\Sigma' = \Sigma^*$,
– $q_0' = (q_0, Z_0)$,
– δ' est le plus petit ensemble inclus dans $Q' \times \Sigma' \times Q'$ tel que: pour tout $\gamma, \beta \in , ^*$, $w \in \Sigma^*$, $Z \in ,$ et $p, q \in Q$:
Si $((p, w, Z), (q, \beta)) \in \delta$ alors $((p, Z\gamma), w, (q, \beta\gamma)) \in \delta'$.

Définition 2.35 (Langage fini et ω-langage d'un automate à pile) :

Soit $\mathcal{A} = (Q, \Sigma, , , \delta, q_0, Z_0)$ un automate à pile et $\mathcal{S}_\mathcal{A} = (Q', \Sigma', q_0', \delta')$ le système de transitions étiquetées associé. Le langage de mots finis de $\mathcal{A}$, noté $L_{fin}(\mathcal{A})$ est l'ensemble des séquences $\sigma \in \Sigma^*$ tel qu'il existe $\sigma' \in (\Sigma^*)^*$ avec $\sigma(0)\sigma(1)\ldots\sigma(|\sigma| \Leftrightarrow 1) = \sigma'(0)\sigma'(1)\ldots\sigma(|\sigma'| \Leftrightarrow 1)$ et σ a un calcul terminant dans $\mathcal{S}_\mathcal{A}$.

Le ω-langage, noté $L(\mathcal{A})$ est l'ensemble de séquences $\sigma \in \Sigma^\omega$ tel qu'il existe $\sigma' \in (\Sigma^*)^\omega$ avec $\sigma(0)\sigma(1)\ldots = \sigma'(0)\sigma'(1)\ldots$ et σ a un calcul dans $\mathcal{S}_\mathcal{A}$.

Remarque: Le langage de mots finis d'un automate à pile est le langage accepté par pile vide. Le ω-langage est donné par toutes les séquences infinies.

Une autre façon de produire des langages de mots finis des automates à pile est donnée par les grammaires hors-contextes. Les grammaires hors-contextes et les automates à pile définissent la même classe de langages de mots finis.

Définition 2.36 (Grammaire hors-contexte) :

Une *grammaire hors-contexte* est un quadruplet $\mathcal{G} = (Var, \Sigma, R, S)$, où

– Var est un alphabet fini constituant les symboles *non-terminaux*,
– Σ est un alphabet fini disjoint de Var, les symboles *terminaux*,

tel-00004890, version 1 - 19 Feb 2004

– $R \subseteq (Var \times (Var \cup \Sigma)^*)$ est un ensemble de *règles*, appelées aussi *productions*. Nous écrivons aussi $V \rightarrow_{\mathcal{G}} \alpha$ pour $(V, \alpha) \in R$
– $S \in Var$ est le *symbole de départ*.

Définition 2.37 :

Soient $\mathcal{G} = (Var, \Sigma, R, S)$ une grammaire et $u \in (Var \cup \Sigma)^+$, $v \in (Var \cup \Sigma)^*$. Alors, nous écrivons $u \Rightarrow_{\mathcal{G}} v$, si $u = xu'y$, $v = xv'y$ et $(u', v') \in R$. La fermeture réflexive et transitive de la relation $\Rightarrow_{\mathcal{G}}$ est représentée par $\Rightarrow^*_{\mathcal{G}}$.

Définition 2.38 (Langage d'une grammaire) :

Soit $\mathcal{G} = (Var, \Sigma, R, S)$ une grammaire. Le langage fini de $\mathcal{G}$, dénoté $L(\mathcal{G})$, est donné par:

$$L(\mathcal{G}) = \{v \in \Sigma^* \ : \ S \Rightarrow^*_{\mathcal{G}} v\}$$

Si nous mettons des conditions d'acceptation sur les calculs infinis du système de transitions étiquetées associé à un automate à pile, nous obtenons un ω-automate à pile [CG77a]. Nous utilisons la condition d'acceptation la plus générale.

Définition 2.39 (ω-automate à pile) :

Un ω-*automate à pile* est un septuplet $\mathcal{A} = (Q, \Sigma, , , \delta, q_0, Z_0, F)$, où

– $(Q, \Sigma, , , \delta, q_0, Z_0)$ est un automate à pile,
– $F \subseteq 2^Q$.

Définition 2.40 (calcul accepteur) :

Soient $\mathcal{A} = (Q, \Sigma, , , \delta, q_0, Z_0, F)$ un ω-automate à pile et $\rho \in (Q, , ^*)^\omega$ un calcul de $\mathcal{S}_{\mathcal{A}}$. Nous écrivons $Inf(\rho)$ pour l'ensemble d'états tel que $\exists^\infty i \in I\!N$ avec $\exists \gamma \in , ^* \rho(i) = (q, \gamma)$. ρ est *accepteur* si $Inf(\rho) \in F$.

Définition 2.41 (ω-langage d'un ω-automate à pile) :

Le ω-*langage* d'un automate à pile $\mathcal{A}$, noté $L(\mathcal{A})$, est l'ensemble des séquences $\sigma \in \Sigma^\omega$ tel que il existe $\sigma' \in (\Sigma^*)^\omega$ avec $\sigma(0)\sigma(1)\ldots = \sigma'(0)\sigma'(1)\ldots$ et $\mathcal{S}_{\mathcal{A}}$ a un calcul accepteur sur σ'.

Définition 2.42 (Langage ω-hors-contexte) :

Un ω-langage accepté par un ω-automate à pile est appelé ω-hors-contexte.

tel-00004890, version 1 - 19 Feb 2004

Les langages ω-hors-contexte ont été étudiés dans [CG77a, CG78, CG77b]. Ils peuvent aussi être définis par les *grammaires ω-hors-contextes*. Nous utilisons les résultats suivants de [CG77a, CG77b]:

Théorème 2.3 :

La classe des langages ω-hors-contexte est fermée par intersection avec les langages ω-réguliers.

Théorème 2.4 :

Soit L un langage ω-hors-contexte. Alors $Pref(L)$ est hors-contexte.

2.5 Les algèbres de processus

Nous définissons l'algèbre de processus PA [BK88] (*Process Algebra)*. Nous donnons la syntaxe de processus PA et définissons leur sémantique opérationnelle. PA est une classe de processus définie par un ensemble de processus atomiques en considérant le choix non-déterministe, la composition séquentielle, la composition parallèle (sans communication) et la récursion. Nous définissons des sous-classes de PA: BPA (*Basic Process Algebra*) qui correspondent aux *processus hors-contexte* et BPP (*Basic Parallel Processes*). BPA ne permet pas la composition parallèle et BPP ne permet pas la composition séquentielle. La classe RP correspond aux processus réguliers qui ont le même pouvoir expressif que les automates finis.

2.5.1 La syntaxe des algèbres PA, BPA, BPP, RP

Nous donnons la définition des algèbres de processus PA (*Process Algebra*), BPA (*Basic Process Algebra*), BPP (*Basic Parallel Processes*) et RP (*Regular Processes*). Soit Σ un alphabet fini et *Var* un ensemble de variables. Nous utilisons les lettres $a, b, \ldots$ pour les éléments de Σ et $X, Y, \ldots$ pour les éléments de *Var*.

Définition 2.43 (Termes de PA) :

L'ensemble $\mathcal{T}$ des termes de PA construit sur *Var* et Σ est défini par la grammaire suivante:

$$t ::= \mathbf{0} \mid a \mid X \mid t + t \mid t \cdot t \mid t \| t$$

Par exemple, $(X\|Y) \cdot ((X\|a) + (X\|b))$ est un terme de PA.

Définition 2.44 (Termes de BPA) :

L'ensemble $\mathcal{T}$ des termes de BPA construit sur *Var* et Σ est défini par la grammaire suivante:

$$t ::= \mathbf{0} \mid a \mid X \mid t + t \mid t \cdot t$$

tel-00004890, version 1 - 19 Feb 2004

Par exemple, $X \cdot a \cdot Z + a$ est un terme de BPA.

Définition 2.45 (Termes de BPP) :

L'ensemble $\mathcal{T}$ des termes de BPP construit sur *Var* et Σ est défini par la grammaire suivante:

$$t ::= \mathbf{0} \mid a \mid X \mid t + t \mid a \cdot t \mid t \| t$$

Par exemple, $(X\|X\|X\|(Z + X))$ est un terme de BPP.

Définition 2.46 (Termes de RP) :

L'ensemble $\mathcal{T}$ des termes de RP construit sur *Var* et Σ est défini par la grammaire suivante:

$$t ::= \mathbf{0} \mid X \mid t + t \mid a \cdot t$$

Intuitivement, $\mathbf{0}$ représente le processus inactif, a est une action atomique, l'opérateur "+" designe le choix non-déterministe, l'opérateur "·" est la composition séquentielle, "$\|$" est la composition parallèle (sans synchronisation).

Définition 2.47 (Processus PA) :

Soit Σ un alphabet fini. Un processus PA $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ est donné par:

- un ensemble fini de variables de processus $Var = \{X_1, \ldots, X_n\}$,
- un ensemble fini Σ d'actions atomiques,
- un ensemble d'équations récursives $\Delta = \{X_i \mathrel{\widehat{=}} t_i \ : \ 1 \leq i \leq n\}$ où chaque terme t_i est un terme de PA sur *Var* et Σ.
- une variable $X_1 \in Var$, appelée la *racine*

Définition 2.48 (Processus BPA,BPP,RP) :

Soit Σ un alphabet fini. Un processus BPA (resp. BPP, RP) $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ est un processus PA tel que tous les termes dans les équations sont des termes de BPA (resp. BPP, RP).

Définition 2.49 (Processus gardé) :

Une occurrence d'une variable X dans un terme t de PA est *gardée* si t comprend un sous-terme $a.t'$, où a est une action atomique et t' contient l'occurrence de X. Un terme t de PA est *gardée* si toutes les occurrences de variables dans t sont gardées. Un processus $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ avec $\Delta = \{X_i \mathrel{\widehat{=}} t_i \ : \ 1 \leq i \leq n\}$ est *gardé* si tous les termes t_i sont gardés.

La notion de processus gardé nous permet de définir une sémantique opérationnelle qui est à branchement fini.

tel-00004890, version 1 - 19 Feb 2004

2.5.2 Sémantique opérationnelle et langages générés

Nous donnons dans cette section la définition de la sémantique opérationnelle des processus ainsi que leur langages associés.

Définition 2.50 (Sémantique opérationnelle d'un processus PA) :

La sémantique opérationnelle d'un processus gardé de PA $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ est donnée par le système de transitions étiquetées $\mathcal{S}' = (\mathcal{T}, \Sigma, X_1, \delta)$ où δ est la plus petite relation telle que:

- $a \xrightarrow{a} \mathbf{0}$
- $X = t \in \Delta$ et $t \xrightarrow{a} t'$ implique $X \xrightarrow{a} t'$,
- $t_1 \xrightarrow{a} t_1'$ implique $t_1 + t_2 \xrightarrow{a} t_1'$,
- $t_1 \xrightarrow{a} t_1'$ implique $t_2 + t_1 \xrightarrow{a} t_1'$,
- $t_1 \xrightarrow{a} t_1'$ implique $t_1 \cdot t_2 \xrightarrow{a} t_1' \cdot t_2$,
- $t_1 \xrightarrow{a} t_1'$ implique $t_1 \| t_2 \xrightarrow{a} t_1' \| t_2$,
- $t_1 \xrightarrow{a} t_1'$ implique $t_2 \| t_1 \xrightarrow{a} t_2 \| t_1'$

Définition 2.51 (langage de PA,BPA,BPP,RP) :

Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA (resp. BPA,BPP,RP) et $\mathcal{S}' = (\mathcal{T}, \Sigma, X_1, \delta)$ le système de transitions correspondant. Le langage fini de $\mathcal{S}$ est donné par:

$$L_{fin}(\mathcal{S}) = \{\sigma \in \Sigma^* \ : \ \text{il existe un calcul terminant } \rho \text{ dans } \mathcal{S}' \text{ sur } \sigma\}$$

Définition 2.52 (ω-langage de PA,BPA,BPP,RP) :

Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA (resp. BPA,BPP,RP) et $\mathcal{S}' = (\mathcal{T}, \Sigma, X_1, \delta)$ le système de transitions correspondant. Le *ω-langage* de $\mathcal{S}$ est donné par:

$$L(\mathcal{S}) = \{\sigma \in \Sigma^\omega \ : \ \text{il existe un calcul sur } \sigma \text{ de } \mathcal{S}'\}$$

Exemple 2.1 :

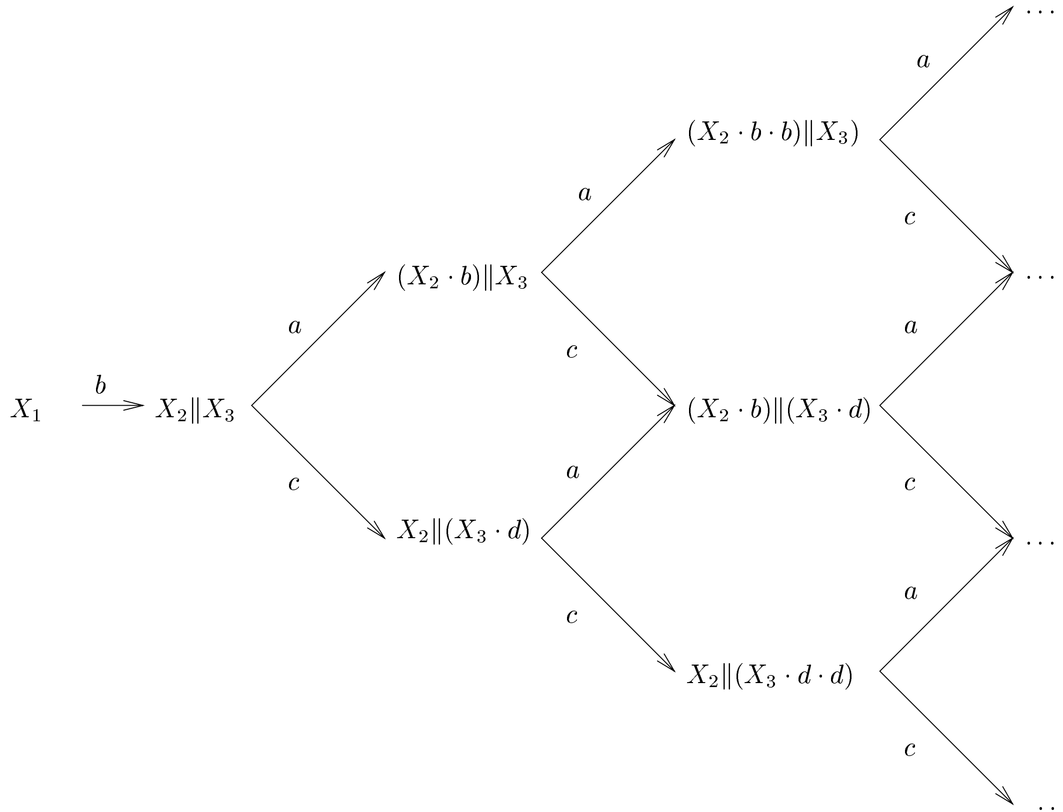
Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA donné par

- $Var = \{X_1, X_2, X_3\}$
- $\Sigma = \{a, b, c, d\}$
- Δ comporte les règles:
  - $X_1 \widehat{=} b \cdot (X_2 \| X_3)$

tel-00004890, version 1 - 19 Feb 2004

- $X_2 \widehat{=} a \cdot X_2 \cdot b$
- $X_3 \widehat{=} c \cdot X_3 \cdot d$

Alors, le "début" du système de transitions étiquetées associé à $\mathcal{S}$ est donné par l'image suivante:



Le langage de $\mathcal{S}$ est $\emptyset$ et le ω-langage de ce processus est $\{a, c\}^\omega$

2.5.3 Forme normale

Nous introduisons une forme spéciale syntaxique de processus PA, qui est une généralisation de la forme normale de Greibach utilisée dans la théorie des langages formels (voir [Har78]).

Définition 2.53 (Termes de la forme normale) :

Soit *Var* un ensemble de variables de processus. L'ensemble $\mathcal{T}_n$ des termes de forme normale construit sur *Var* est défini par la grammaire suivante:

$$t ::= \mathbf{0} \mid X \mid t \cdot t \mid t \| t$$

tel-00004890, version 1 - 19 Feb 2004

Définition 2.54 (Forme normale de PA) :

Un processus PA $(Var, \Sigma, \Delta, X_1)$ avec $\Delta = \{X_i \hat{=} t_i \ : \ 1 \leq i \leq n\}$ est en *forme normale*, si chaque terme t_i est soit $\mathbf{0}$ soit de la forme

$$a_1^i \cdot \tau_1^i + \cdots + a_{m_i}^i \cdot \tau_{m_i}^i$$

où $a_1^i, \ldots, a_{m_i}^i \in \Sigma$ et $\tau_1^i, \ldots, \tau_{m_i}^i \in \mathcal{T}_n$ et telle qu'aucune variable X_k dans un τ_j^i est définie comme un processus inactif, c.-à-d.. $X_k \hat{=} \mathbf{0} \in \Delta$.

Proposition 2.1 (Forme normale de processus PA) :

Soit $(Var, \Sigma, \Delta, X_1)$ avec $\Delta = \{X_i \hat{=} t_i \ : \ 1 \leq i \leq n\}$ un processus PA gardé. Alors il existe un processus PA $\mathcal{S}_1$ en forme normale, tel que les systèmes de transitions définissent les mêmes ω-langages, i.e. $L(\mathcal{S}) = L(\mathcal{S}_1)$

Preuve:
Par induction sur la structure des termes et en transformant les termes par rapport aux équations suivantes

$$\begin{aligned}
(t_1 + t_2) \cdot t_3 &= t_1 \cdot t_3 + t_2 \cdot t_3 && (2.1)\\
t_3 \cdot (t_1 + t_2) &= t_3 \cdot t_1 + t_3 \cdot t_2 && (2.2)\\
(t_1 + t_2) \| t_3 &= (t_1 \| t_3) + (t_2 \| t_3) && (2.3)\\
t_3 \| (t_1 + t_2) &= (t_3 \| t_1) + (t_3 \| t_2) && (2.4)
\end{aligned}$$

qui sont valides modulo équivalences des langages, c'est-à-dire deux termes liés par ces équations génèrent les mêmes langages. Avec ces équations les termes peuvent être transformés de sorte que les opérateurs + sont au premier niveau. Puisque le processus est gardé nous pouvons obtenir la forme exigée. □

De la même manière nous pouvons définir des formes normales pour les processus BPA et BPP.

Exemple 2.2 :

La forme normale du processus de l'exemple 2.1 est donnée par les équations:

- $X_1 \hat{=} b \cdot (X_2 \| X_3)$
- $X_2 \hat{=} a \cdot X_2 \cdot X_4$
- $X_3 \hat{=} c \cdot X_3 \cdot X_5$
- $X_4 \hat{=} b$
- $X_5 \hat{=} d$

tel-00004890, version 1 - 19 Feb 2004

2.5.4 Notations et résultats préliminaires

Nous introduisons dans cette section quelques notations et des résultats préliminaires concernant les algèbres de processus.

Définition 2.55 :

Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ en forme normale. Nous définissons l'ensemble de règles de transitions suivant:

$$\begin{array}{rcl} \rho_{\mathcal{S}} & = & \{X_i \rightarrow a^i_j.\tau^i_j : X_i \widehat{=} a^i_1 \cdot \tau^i_1 + \cdots + a^i_j \cdot \tau^i_j + \cdots + a^i_{m_i} \cdot \tau^i_{m_i} \in \Delta\} \\ & & \cup \{X_i \rightarrow \mathbf{0} : X_i \widehat{=} \mathbf{0} \in \Delta\} \end{array}$$

Définition 2.56 (Positions) :

Soit $\tau \in \mathcal{T}$. L'ensemble de toutes les *positions* de τ est un sous-ensemble de $I\!N^*$ inductivement défini comme suit:

$$\begin{array}{lcl} Pos(\mathbf{0}) & = & \emptyset, \\ Pos(X) & = & \{\Lambda\}, \text{ où } \Lambda \text{ est la suite vide,} \\ Pos(\tau_1 \ op\ \tau_2) & = & \{1 \cdot p : p \in Pos(\tau_1)\} \cup \{2 \cdot p : p \in Pos(\tau_2)\} \\ & & \text{où } op \text{ est } \| \text{ ou } \cdot \end{array}$$

Définition 2.57 (Position exécutable) :

L'ensemble de *positions exécutables* est défini inductivement sur les termes comme suit:

$$\begin{array}{lcl} Exec(\mathbf{0}) & = & \emptyset, \\ Exec(X) & = & \{\Lambda\}, \\ Exec(\tau_1 \cdot \tau_2) & = & \{1 \cdot p : p \in Exec(\tau_1)\}, \\ Exec(\tau_1 \| \tau_2) & = & \{1 \cdot p : p \in Exec(\tau_1)\} \cup \{2 \cdot p : p \in Exec(\tau_2)\}. \end{array}$$

Nous disons qu'une variable X est exécutable dans le terme τ s'il y a une position $p \in Exec(\tau)$ telle que le sous-terme de τ à la position p est X. Nous notons $Var_{Exec}(\tau)$ l'ensemble de toutes les variables exécutables dans τ.

La notion de variable exécutable nous permet de définir une relation de transition entre termes de $\mathcal{T}$ en utilisant des règles de transition dans $\delta_{\mathcal{S}}$. Ces règles peuvent être interprétées comme des règles de réécriture.

Soient $\tau \in \mathcal{T}$, X une variable avec $X \in Var_{Exec}(\tau)$, p une position exécutable de τ et une règle $r = X \rightarrow a.\alpha \in \rho_{\mathcal{S}}$. Alors nous écrivons $\tau \stackrel{a}{\rightarrow}_r \tau'$, où τ' est le terme où la variable X à la position p a été remplacé par le terme α. Nous généralisons cette notation à des séquences de transitions en écrivant $\tau \stackrel{\sigma}{\rightarrow}_{\mathcal{S}} \tau'$, ou $\sigma = a_1 \cdots a_n \in \Sigma^+$ si $\tau \stackrel{a_1}{\rightarrow}_{r_1} \tau_1 \cdots \stackrel{a_n}{\rightarrow}_{r_n} \tau_n = \tau'$ et $\{r_1, \cdots, r_n\} \subseteq \rho_{\mathcal{S}}$. Une séquence $\tau \stackrel{a_1}{\rightarrow}_{\mathcal{S}} \tau_1 \cdots \stackrel{a_n}{\rightarrow}_{\mathcal{S}} \tau_n$ correspond exactement à la même séquence dans le système de transition associé à $\mathcal{S}$.

Définition 2.58 (variables normées) :

Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA. L'ensemble de *variables normées* qui peuvent se réduire à $\mathbf{0}$ est défini par:

$$Norm_{\mathcal{S}} = \{X \in Var : \exists \sigma \in \Sigma^+ . X \stackrel{\sigma}{\rightarrow}_{\mathcal{S}} \mathbf{0}\}$$

tel-00004890, version 1 - 19 Feb 2004

Nous pouvons calculer cet ensemble d'une manière itérative. Toutes les variables avec $X \xrightarrow{a}_{\mathcal{S}} \mathbf{0}$ sont dans $Norm_{\mathcal{S}}$ et si pour une variables X' avec $X' \xrightarrow{a}_{\mathcal{S}} \tau$, toutes les variables de τ sont dans $Norm_{\mathcal{S}}$ alors X est aussi dans $Norm_{\mathcal{S}}$.

Nous considérons au chapitre 4 la vérification de processus PA par rapport à des logiques. Pour cela nous supposons que $\Sigma = 2^{\mathcal{P}}$ où $\mathcal{P}$ est un ensemble fini de propositions atomiques. Soit π une formule propositionnelle sur $\mathcal{P}$. Nous définissons l'ensemble de toutes les variables qui peuvent se réduire à $\mathbf{0}$ en satisfant continûment π.

Définition 2.59 :

Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA. L'ensemble de *variables normées* qui peuvent se réduire à $\mathbf{0}$ en satisfaisant continûment π est défini par:

$$Norm^{\pi}_{\mathcal{S}} = \{\tau \in \mathcal{T} : \exists \sigma \in \Sigma^{+}.\tau \xrightarrow{\sigma}_{\mathcal{S}} \mathbf{0} \text{ et } \forall i \in \{0, \ldots, |\sigma| \Leftrightarrow 1\},\ \sigma(i) \models \pi\}$$

Cet ensemble peut être calculer de la même manière que $Norm_{\mathcal{S}}$.

Définition 2.60 :

Soient $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA et π une formule d'états, alors l'ensemble de variable de processus qui peuvent répéter infiniment souvent π est définie comme suit:

$$\begin{aligned} Boucle^{\pi}_{\mathcal{S}} \quad = \quad & \{X \in Var_{\Delta} \ : \ \exists \sigma \in \Sigma^{+}, \text{ et } \exists \tau, \text{tel que } X \xrightarrow{\sigma}_{\mathcal{S}} \tau, \\ & X \in Var_{Exec}(\tau), \text{ et } \forall i \in \{0, \ldots, |\sigma| \Leftrightarrow 1\},\ \sigma(i) \models \pi\} \end{aligned}$$

Intuitivement, si $X \in Boucle^{\pi}_{\mathcal{S}}$ alors il existe une séquence infini à partir de X qui satisfait continûment π.

Nous pouvons calculer l'ensemble $Boucle^{\pi}_{\mathcal{S}}$ comme l'ensemble de nonterminaux recursives dans une grammaire hors-contexte en utilisant l'ensemble $Norm^{\pi}_{\mathcal{S}}$.

Nous terminons cette section avec un résultat préliminaire sur les langages de PA.

Lemme 2.9 :

Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA en forme normale. Alors il existe un processus $\mathcal{S}' = (Var', \Sigma, \Delta', X'_1)$ tel que $L_{fin}(\mathcal{S}') = \emptyset$ et $L(\mathcal{S}) = L(\mathcal{S}')$.

tel-00004890, version 1 - 19 Feb 2004

Preuve:
La construction du processus $\mathcal{S}'$ est très simple. Nous devons assurer que dans chaque terme dans le système de transition décrit par $\mathcal{S}'$ il y a exactement une variable qui ne peut pas terminer (n'est pas inclus dans $Norm_{\mathcal{S}'}$). Pour chaque variable X nous introduisons une copie X^t qui ne peut pas terminer et nous changeons les règles de sorte que dans le systèmes de transitions associé à $\mathcal{S}'$ chaque terme contient exactement une variable de la forme X^t. Formellement, $\mathcal{S}' = (\mathit{Var}', \Sigma, \Delta', X_1')$ est donné par:

- Si $\mathit{Var} = \{X_1, \ldots, X_n\}$ alors $\mathit{Var}' = \mathit{Var} \cup \{X_1^t, \ldots, X_n^t\}$
- Δ' est l'ensemble plus petit qui satisfait:
  - $\Delta \subseteq \Delta'$,
  - Si $X \mathrel{\widehat{=}} \sum_{i=1}^{n} a_i \cdot \tau_i \in \Delta$ alors
    $X^t \mathrel{\widehat{=}} \sum_{\tau_i \neq \mathbf{0}} (\sum_{\tau \in T(\tau_i)} a_i \cdot \tau) \in \Delta'$ où $T(\tau_i)$ est l'ensemble de termes où exactement une des variables X de τ_i a été remplacé par X^t
- $X_1' = X_1^t$

Il est évident que $L_{fin}(\mathcal{S}') = \emptyset$ et $L(\mathcal{S}) = L(\mathcal{S}')$. □

2.6 Les réseaux de Petri

Dans cette section nous définissons les réseaux de Petri et un modèle équivalent: les systèmes d'addition de vecteurs avec états (SAVE). Ces définitions peuvent être trouvées par exemple dans [Reu89]. Nous mentionnons ensuite un résultat de Christensen qui a montré dans [Chr93] que l'algèbre de processus BPP est une sous-classe des réseaux de Petri.

2.6.1 Les réseaux de Petri

Définition 2.61 (réseau de Petri) :

Un *réseau de Petri* est un sextuplet $\mathcal{N} = (\Sigma, \mathbf{P}, \mathbf{T}, \mathbf{F}, M_{\mathcal{N}}, \Lambda)$ où,

- Σ est un alphabet fini,
- $\mathbf{P}$ est un ensemble fini de *places*,
- $\mathbf{T}$ est un ensemble fini de *transitions*,
- $\mathbf{P} \cap \mathbf{T} = \emptyset$,
- $\mathbf{F} : (\mathbf{P} \times \mathbf{T}) \cup (\mathbf{T} \times \mathbf{P}) \rightarrow I\!N$ est la *fonction de flot*,

tel-00004890, version 1 - 19 Feb 2004

- $M_{\mathcal{N}} : \mathbf{P} \rightarrow I\!N$ est le *marquage initial* et $\Lambda : \mathbf{T} \rightarrow \Sigma$ est la *fonction d'étiquetage* des transitions.

Définition 2.62 (Marquage, Tir d'une transition) :

Soit $\mathcal{N} = (\Sigma, \mathbf{P}, \mathbf{T}, \mathbf{F}, M_{\mathcal{N}}, \Lambda)$. Un marquage M est une fonction qui associe à chaque place un entier (nombre de *jeton*). Nous représentons un marquage aussi comme un vecteur dans $I\!N^{|\mathbf{P}|}$. Nous écrivons $M[t\rangle$ si, $\forall p \in P$, $M(p) \geq \mathbf{F}(p,t)$, et nous écrivons $M[t\rangle M'$ (t est *tiré* à partir de M et donne M'), si $M[t\rangle$ et $\forall p \in P$, $M'(p) = M(p) \Leftrightarrow \mathbf{F}(p,t) + \mathbf{F}(t,p)$. Étant donné un $a \in \Sigma$, nous écrivons $M \stackrel{a}{\rightarrow}$ (resp. $M \stackrel{a}{\rightarrow} M'$) si $\exists t \in \mathbf{T}$ tel que $M[t\rangle$ (resp. $M[t\rangle M'$) et $\Lambda(t) = a$.

La définition de $M[t\rangle M'$ (resp. $M \stackrel{a}{\rightarrow} M'$) est étendue d'une manière évidente vers des séquences de transitions dans $\tau \in \mathbf{T}^*$ (respectivement des séquences d'étiquettes de transition $\sigma \in \Sigma^*$). La définition $M[t\rangle$ (respectivement $M \stackrel{a}{\rightarrow}$) est étendue d'une manière évidente vers des séquences de transitions dans $\tau \in \mathbf{T}^\omega$ (resp. des séquences d'étiquettes de transition $\sigma \in \Sigma^\omega$).

Définition 2.63 (Ensemble de marquages atteignables) :

Soit $\mathcal{N} = (\Sigma, \mathbf{P}, \mathbf{T}, \mathbf{F}, M_{\mathcal{N}}, \Lambda)$. L'ensemble de marquages atteignables $\mathcal{R}(\mathcal{N})$ est donné par:

$$\mathcal{R}(\mathcal{N}) = \{M \;:\; \exists \tau \in \mathbf{T}^*.\ M_{\mathcal{N}}[\tau\rangle M\}$$

Définition 2.64 (ω-langage d'un réseau de Petri) :

Soit $\mathcal{N} = (\Sigma, \mathbf{P}, \mathbf{T}, \mathbf{F}, M_{\mathcal{N}}, \Lambda)$. Le ω-langage de $\mathcal{N}$, représenté par $L(\mathcal{N})$, est donné par:

$$L(\mathcal{N}) = \{\sigma \in \Sigma^\omega : M_{\mathcal{N}} \stackrel{\sigma}{\rightarrow}\}$$

Définition 2.65 ($\mathcal{M}_\infty(\mathcal{N}, t)$) :

Soit $\mathcal{N} = (\Sigma, \mathbf{P}, \mathbf{T}, \mathbf{F}, M_{\mathcal{N}}, \Lambda)$. Étant donné un $t \in \mathbf{T}$, nous écrivons $\mathcal{M}_\infty(\mathcal{N}, t)$ pour l'ensemble de marquage M telle que $\exists \tau \in \mathbf{T}^\omega$ avec $M[\tau\rangle$ et $\exists^\infty i \in I\!N$ avec $\tau(i) = t$, i.e. l'ensemble de tous les marquages à partir desquels il existe une séquence infinie de transitions qui contient infiniment souvent t.

Nous avons le théorème suivant prouvé indépendemment par [Kos82] et [May81].

Théorème 2.5 (Atteignabilité) :

Soit $\mathcal{N} = (\Sigma, \mathbf{P}, \mathbf{T}, \mathbf{F}, M_{\mathcal{N}}, \Lambda)$ un réseau de Petri et M un marquage. Alors, le problème

$$M \in \mathcal{R}(\mathcal{N})?$$

est décidable.

tel-00004890, version 1 - 19 Feb 2004

Nous terminons cette section avec un résultat sur les ensembles semilinéaires de marquages.

Théorème 2.6 :

> Soit $\mathcal{N} = (\Sigma, \mathbf{P}, \mathbf{T}, \mathbf{F}, M_{\mathcal{N}}, \Lambda)$ et $\mathcal{M}$ un ensemble semilinéaire de marquage. Alors, le problème $\mathcal{R}(\mathcal{N}) \cap \mathcal{M} \neq \emptyset$ est décidable.

Preuve:

- D'abord nous pouvons montrer que le problème si $\mathcal{R}(\mathcal{N}_1) \cap \mathcal{R}(\mathcal{N}_2) \neq \emptyset$ pour deux réseaux de Petri $\mathcal{N}_1$ et $\mathcal{N}_2$ avec le même nombre de places est décidable. En effet, ce problème peut être réduit au problème d'atteignabilité dans les réseaux de Petri (qui est décidable, voir Théorème 2.5): D'abord nous pouvons ajouter à $\mathcal{N}_1$ et $\mathcal{N}_2$ des transitions qui enlèvent simultanément un par un les jetons des places avec le même indice. Alors, $\mathcal{R}(\mathcal{N}_1) \cap \mathcal{R}(\mathcal{N}_2) \neq \emptyset$ ssi le marquage vide est atteignable dans le réseau ainsi construit: Si $\mathcal{R}(\mathcal{N}_1) \cap \mathcal{R}(\mathcal{N}_2) \neq \emptyset$, alors on peut atteindre le marquage vide trivialement. Si le marquage vide est atteignable dans le réseau construit, alors il existe une séquence de transitions qui l'atteint. Cette séquence peut toujours être réarrangée telle que les nouvelles transitions qui vident les places sont exécutées à la fin. Il s'en suit qu'il y a un marquage dans $\mathcal{R}(\mathcal{N}_1) \cap \mathcal{R}(\mathcal{N}_2)$.
- Soit $\mathcal{L}$ un ensemble linéaire. Nous pouvons facilement construire un réseau de Petri $\mathcal{N}_{\mathcal{L}}$ avec $\mathcal{R}(\mathcal{N}_{\mathcal{L}}) = \mathcal{L}$.

Le problème $\mathcal{R}(\mathcal{N}) \cap \mathcal{L} \neq \emptyset$ pour un ensemble linéaire $\mathcal{L}$ peut donc être réduit vers le problème d'atteignabilité. Puisque un ensemble semilinéaire est une union finie d'ensembles linéaires ce raisonnement peut facilement être généralisé. □

2.6.2 Les systèmes d'addition de vecteurs avec états

Nous utilisons un modèle équivalent aux réseaux Petri. La définition des systèmes d'addition de vecteurs avec états peut par exemple être trouvée dans [Reu89].

Définition 2.66 (Vecteurs) :

> Pour un vecteur $\vec{u} \in \mathbb{Z}^k$ et un i avec $1 \leq i \leq k$, nous écrivons $\vec{u}(i)$ pour la i-ème composante de $\vec{u}$. $\vec{u}(i)$ est aussi appelé i-ème place.

Définition 2.67 (Opérations sur les vecteurs) :

> Soient $\vec{u}$,$\vec{v} \in \mathbb{Z}^k$. L'addition $\vec{u} + \vec{v}$, la soustraction $\vec{u} \Leftrightarrow \vec{v}$ et les prédicats $\vec{u} = \vec{v}$, $\vec{u} \geq \vec{v}$ et $\vec{u} < \vec{v}$ sont définies comme d'habitude par composante. $\vec{0}$ dénote le vecteur dont toutes les composantes sont 0.

tel-00004890, version 1 - 19 Feb 2004

Définition 2.68 (SAVE) :

Un *système d'addition de vecteurs avec états* à k dimensions étiqueté est un 6-uplet $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$, où

- Σ est un *alphabet* fini,
- $\vec{v}_0 \in \mathbb{N}^k$ est le *vecteur de début*,
- A est un ensemble de vecteur de $\mathbb{Z}^k$ (l'ensemble d'addition),
- Q est un ensemble d'*états de contrôle*,
- $q_0 \in Q$ est *l'état initial*,
- $\delta \subseteq Q \times \Sigma \times Q \times A$ est la *relation de transition*.

Une *transition* $(q_1, b, q_2, \vec{a}) \in \delta$ est normalement écrit dans la forme $q_1 \xrightarrow{b} (q_2, \vec{a})$ (si le b nous n'intéresse pas nous écrivons aussi $q_1 \rightarrow (q_2, \vec{a})$). Une *configuration* d'un SAVE est un couple $(q, \vec{v})$, où $q \in Q$ et $\vec{v} \in \mathbb{N}^k$. La *configuration initiale* est le couple $(q_0, \vec{v}_0)$. Une configuration $(q_1, \vec{v})$ peut tirer une transition $q_1 \xrightarrow{b} (q_2, \vec{a})$ en donnant la configuration $(q_2, \vec{v} + \vec{a})$ si $\vec{v} + \vec{a} \geq \vec{0}$. Nous notons cela $(q_1, \vec{v}) \overset{b}{\rightsquigarrow} (q_2, \vec{v} + \vec{a})$ (où $(q_1, \vec{v}) \rightsquigarrow (q_2, \vec{v} + \vec{a})$). Cette définition peut être généralisé d'une manière évidente pour des séquences de Σ^∞. Nous écrivons $\rightsquigarrow^+$ (respectivement $\rightsquigarrow^*$), pour la fermeture transitive (resp. transitive et réflexive) de $\rightsquigarrow$.

Définition 2.69 (ω langage d'un SAVE) :

Le ω-langage d'un SAVE $\mathcal{S}$, appelé $L(\mathcal{S})$, est défini par l'ensemble des séquences infinies $\sigma \in \Sigma^\omega$ tel que $(q_0, \vec{v}_0) \overset{\sigma}{\rightsquigarrow}$.

Il est clair que pour chaque SAVE à k dimension et n états, il y a un réseau de Petri équivalent (équivalent dans le sens qu'il décrit le même système à transitions étiqueté) à $k+n$ dimensions (chaque état correspond à une nouvelle place). Et chaque réseau de Petri peut être définie par un SAVE. Nous utilisons les SAVE dans le chapitre 3 pour avoir une représentation adéquate pour le produit d'un automate de Büchi et d'un réseau de Petri.

2.6.3 Réseau BPP et SAVE

Les processus BPP peuvent être vus comme une sous-classe des réseaux de Petri.

Définition 2.70 :

Un réseau BPP est un SAVE $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ tel que $Q = \{q_0\}$ et A ne contient que de vecteurs qui ont une seule composante négative. Cette composante doit avoir la valeur -1.

Christensen a montré dans [Chr93] que les réseaux BPP et les processus BPP définissent la même classe de système de transitions étiquetés.

Lemme 2.10 :

Les réseaux BPP et les processus BPP définissent la même classe de système à transitions étiquetés.

tel-00004890, version 1 - 19 Feb 2004

2.7 Semilinéarité

Dans cette section nous définissons les systèmes semilinéaires. Les systèmes semilinéaires sont des systèmes qui génèrent des ω-langages dont l'image de Parikh des préfixes est semilinéaire. Cette propriété nous permet de raisonner facilement sur les nombres d'occurrences de symboles dans ces systèmes (voir chapitre 4). Nous donnons des exemples de systèmes semilinéaires et montrons des résultats les concernant.

2.7.1 Définitions

Dans cette section nous définissons les systèmes semilinéaires. Ce sont les systèmes dont les images de Parikh des préfixes de leurs ω-mots sont semilinéaires. Soient Σ un alphabet fini et $\sigma \in \Sigma^*$. Pour chaque $a \in \Sigma$, $|\sigma|_a$ est le nombre d'occurrences de a dans σ.

Définition 2.71 (Image de Parikh) :

Soit $\Sigma = \{a_1, \ldots, a_n\}$. L' image de Parikh de $\sigma \in \Sigma^*$, représentée par $[\sigma]$, est le vecteur $(|\sigma|_{a_1}, \cdots, |\sigma|_{a_n})$ de $I\!N^n$. Soit $S \subseteq \Sigma^*$, alors $[S] = \{[\sigma] \ : \ \sigma \in S\}$.

Définition 2.72 (Langage semilinéaire) :

Un ensemble $S \subseteq \Sigma^*$ est un *langage semilinéaire* si $[S]$ est semilinéaire.

Définition 2.73 (ω-langage semilinéaire) :

Un ensemble $S \subseteq \Sigma^\omega$ est un *ω-langage semilinéaire* si $[Pref(S)]$ est semilinéaire.

Définition 2.74 (Système semilinéaire) :

Un système est semilinéaire si son ω-langage S est semilinéaire et tel qu'une représentation de $[Pref(S)]$ peut être construite à partir d'une représentation de S.

2.7.2 Résultats

Dans cette section nous présentons quelques résultats concernant les systèmes semilinéaires. Nous montrons que les langages hors-contextes et ω-hors-contextes sont semilinéaires. Par conséquent les processus BPA et les automates à pile sont des systèmes semilinéaires. Nous montrons aussi que les processus PA sont semilinéaires. Ensuite nous montrons que la classe des systèmes semilinéaires n'est pas fermée par intersection.

Le théorème suivant est du à Parikh [Par66].

Théorème 2.7 :

Les langages hors-contextes sont semilinéaires.

tel-00004890, version 1 - 19 Feb 2004

En utilisant le Théorème 2.4 on déduit:

Théorème 2.8 :

Les langages ω-hors-contextes sont semilinéaires.

Du fait que BPA et les automates à pile génèrent des langages ω-hors-contextes nous obtenons:

Corollaire 2.2 :

Les processus BPA et les automates à pile sont semilinéaires.

Nous prouvons maintenant que les processus PA sont semilinéaires

Théorème 2.9 :

Les ω-langages de PA sont semilinéaires.

Preuve:
Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA. Puisque nous considérons ici le langage produit par $\mathcal{S}$ nous pouvons supposer que $\mathcal{S}$ est en forme normale. Nous supposons aussi que le langage *fini* de $\mathcal{S}$ est vide (Lemme 2.9). La construction de $[Pref(L(\mathcal{S}))]$ se fait en deux partie:

- Construction d'un processus PA $\mathcal{S}_{fin}$ avec $L_{fin}(\mathcal{S}_{fin}) = Pref(L(\mathcal{S}))$,
- Construction d'une grammaire hors-contexte $\mathcal{G}$ avec $[L(\mathcal{G})] = [L_{fin}(\mathcal{S}_{fin})]$.

D'abord nous construisons le processus $\mathcal{S}_{fin} = (Var_{fin}, \Sigma, \Delta_{fin}, X_{fin})$. Pour chaque variable X nous introduisons une copie X^p de cette variable qui peut produire les préfixes des séquences produite par X. Ensuite nous ajoutons des règles pour ces variables qui permettent de produire les préfixes. Par exemple pour $X \mathrel{\widehat{=}} a \cdot Y \cdot Z$ nous ajoutons $X^p = a + a \cdot Y^p + a \cdot Y \cdot Z^p$. Formellement:

- Si $Var = \{X_1, \ldots, X_n\}$ alors $Var_{fin} = Var \cup \{X_1^p, \ldots, X_n^p\}$,
- Δ_{fin} est le plus petit ensemble qui satisfait:
  - $\Delta \subseteq \Delta_{fin}$
  - Si $X \mathrel{\widehat{=}} \sum_{i=1}^n a_i \cdot \tau_i \in \Delta$ alors
    $X^p \mathrel{\widehat{=}} \sum_{i=1}^n (a_i + \sum_{\tau \in Pr(\tau_i)} a_i \cdot \tau) \in \Delta_{fin}$ où Pr est récursivement définie comme suit:

$$\begin{array}{rcl}
Pr(X) & = & \{X^p\} \\
Pr(\tau_1 \cdot \tau_2) & = & \{\tau_1' \ : \ \tau_1' \in Pr(\tau_1)\} \cup \{\tau_1 \cdot \tau_2' \ : \ \tau_2' \in Pr(\tau_2)\} \\
Pr(\tau_1 \| \tau_2) & = & \{\tau_1' \ : \ \tau_1' \in Pr(\tau_1)\} \\
 & & \cup \{\tau_2' \ : \ \tau_2' \in Pr(\tau_1)\} \\
 & & \cup \{\tau_1' \| \tau_2 \ : \ \tau_1' \in Pr(\tau_1)\} \\
 & & \cup \{\tau_1 \| \tau_2' \ : \ \tau_2' \in Pr(\tau_2)\} \\
 & & \cup \{\tau_1' \| \tau_2' \ : \ \tau_1' \in Pr(\tau_1) \wedge \tau_2' \in Pr(\tau_2)\}
\end{array}$$

tel-00004890, version 1 - 19 Feb 2004

- $X_{fin} = X_1^P$

La grammaire hors-contexte $\mathcal{G} = (Var_{\mathcal{G}}, \Sigma, R, S)$ est construite en remplaçant toutes les compositions parallèles dans le processus $\mathcal{S}_{fin}$ par des compositions séquentielles. Cela ne change pas l'image de Parikh d'un mot généré. Formellement:

- $Var_{\mathcal{G}} = Var_{fin}$
- R est le plus petit ensemble qui satisfait:
  - Si $X \,\widehat{=}\, \sum_{i=1}^{n} a_i \cdot \tau_i \in \Delta_{fin}$ alors
    $\forall i \in \{1, \ldots, n\}.\ X \Rightarrow_{\mathcal{G}} a_i Str(\tau_i)$ où Str est définie récursivement comme suit:
    - $Str(\mathbf{0}) = \epsilon$
    - $Str(X) = X$
    - $Str(\tau_1 \cdot \tau_2) = Str(\tau_1 \| \tau_2) = Str(\tau_1) Str(\tau_2)$
- $S = X_{fin}$

Il est évident que $[L(\mathcal{G})] = [L_{fin}(\mathcal{S}_{fin})]$. D'après le Théorème 2.7 $[L(\mathcal{G})]$ est semilinéaire. □

Pour les ω-langages de processus BPP nous pouvons montrer que leur intersection avec un ω-langage sans hauteur d'étoile peut donner un langage qui n'est pas semilinéaire. Cela montre que la classe des *systèmes* semilinéaires n'est pas fermée par intersection bien que les *ensembles* semilinéaires le sont.

Proposition 2.2 :

Il existe un processus BPP $\mathcal{S}$ avec 4 variables et un ω-langage sans hauteur d'étoile R descriptible par un automate avec 2 états, tels que $[Pref(L(\mathcal{S}) \cap R)]$ n'est pas semilinéaire.

Preuve:
La preuve est inspiré d'un exemple dans [HP79] d'un système d'addition de vecteurs avec états à 3 dimensions qui a un ensemble d'états atteignables qui est non-semilinéaire.

Soit $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus BPP avec:

- $Var = \{X_1, X_2, X_3, X_4\}$,
- $\Sigma = \{t_1, t_2, t_3, t_4, init\}$,
- Δ est l'ensemble des cinq équations suivantes:
  - $X_1 \,\widehat{=}\, init \cdot X_2 \| X_3$
  - $X_2 \,\widehat{=}\, t_1 \cdot X_4$
  - $X_3 \,\widehat{=}\, t_2 \cdot X_3$

tel-00004890, version 1 - 19 Feb 2004

- $X_3 \widehat{=} t_4 \cdot X_3$
- $X_4 \widehat{=} t_3 \cdot (X_2 \| X_2)$

Soit R le langage donné par l'automate fini B avec états p et q, état initial q, vocabulaire $\{t_1, t_2, t_3, t_4, init\}$ et transitions

- $p \stackrel{init}{\rightarrow} p$,$p \stackrel{t_1}{\rightarrow} p$,$p \stackrel{t_2}{\rightarrow} q$,$q \stackrel{t_3}{\rightarrow} q$ et $q \stackrel{t_4}{\rightarrow} p$.

Remarque: Le langage $L(\mathcal{S}) \cap L(B)$ peut être vu comme le langage produit par le SAVE (voir définition 2.68) donné dans figure 2.1.

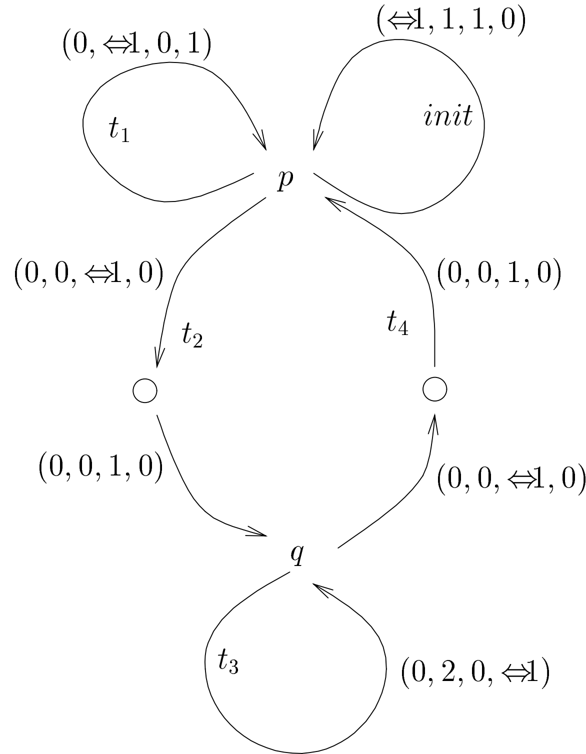


FIG. 2.1 – *Un SAVE qui produit un langage non-semilinéaire*

Soit L l'ensemble de séquences $Pref(L(\mathcal{S}) \cap L(B))$. Nous montrons que $[L]$ n'est pas semilinéaire. Pour $\sigma \in L$ soit $a_i = |\sigma|_{t_i}$ et $in = |\sigma|_{init}$. Soient

- $P(a_2, a_3, a_4) \equiv (a_2 = a_4) \wedge (a_3 + 1 \leq 2^{a_4})$
- $Q(a_1, a_2, a_4) \equiv (a_2 = a_4 + 1) \wedge (a_1 + 1 \leq 2^{a_4+1})$
- $R(a_1, a_2, a_3, a_4, in) \equiv$
  $(in = 0) \wedge ((in = 1) \ \wedge (0 \leq a_3 \leq a_1 \leq 2a_3 + 1) \wedge (a_4 \geq 0) \wedge$
  $(P(a_2, a_3, a_4) \vee Q(a_1, a_2, a_4)))$

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Alors nous avons le lemme suivant à partir duquel nous pouvons déduire que $[L]$ n'est pas semilinéaire parce que les relations dans R contiennent des fonctions exponentielles.

Lemme 2.11 :

$$(a_1, a_2, a_3, a_4, in) \in [L] \iff R(a_1, a_2, a_3, a_4, in)$$

La partie "$\Rightarrow$" est facilement prouvé en montrant qu'après un pas $R(0, 0, 0, 0, 1)$ est vrai et que $R(a_1, a_2, a_3, a_4, in)$ est un invariant par rapport à l'application simultanée des transitions (règles) de Δ et B. Pour cela il faut aussi raisonner sur la forme des états atteignables. Considérons maintenant la partie "$\Leftarrow$". Elle est montrée par induction sur a_4.

- $a_4 = 0$:
  Si $a_4 = 0$ alors si $a_2 = 0$ alors $a_3 = 0$ et $a_1 = 0$ ou $a_1 = 1$. Nous avons $(0, 0, 0, 0, 1) \in [L]$ ainsi que $(1, 0, 0, 0, 1) \in [L]$ par $X_1 \overset{init}{\to} X_2 \| X_3 \overset{t_1}{\to} X_4 \| X_3$ et $p \overset{init}{\to} p \overset{t_1}{\to} p$.
  Si $a_2 = 1$ alors aussi $a_1 = 0$ ou $a_1 = 1$ et nous avons $(0, 1, 0, 0, 1) \in [L]$ par $X_1 \overset{init}{\to} X_2 \| X_3 \overset{t_2}{\to} X_2 \| X_3$ et $p \overset{init}{\to} p \overset{t_2}{\to} q$ ainsi que $(1, 1, 0, 0, 1) \in [L]$ par $X_1 \overset{init}{\to} X_2 \| X_3 \overset{t_1}{\to} X_4 \| X_3) \overset{t_2}{\to} X_4 \| X_3$ et $p \overset{init}{\to} p \overset{t_1}{\to} p \overset{t_2}{\to} q$.

- $a_4 = n$:
  Nous supposons que pour tout $a_4 < n$, $R(a_1, a_2, a_3, a_4, in)$ implique $(a_1, a_2, a_3, a_4, in) \in [L]$.
  Il y a deux cas:

  1. $a_2 = a_4$:
     Soit $(v_1, \ldots, v_5)$ un vecteur qui satisfait R.
     - $0 < v_3 + 1 \leq 2^{v_4 - 1}$:
       En utilisant l'hypothèse d'induction nous savons que $(v_1, v_2 \Leftrightarrow 1, v_3, v_4 \Leftrightarrow 1) \in [L]$. Il existe donc un calcul: $X_1 \overset{init}{\to} X_2 \| X_3 \to^* (X_2^k \| X_4^j \| X_3)$ (avec $k, j \geq 0$) et un calcul $p \overset{init}{\to} \to^* p$ tel que le deux produisent une séquence σ avec $[\sigma] = (v_1, v_2 \Leftrightarrow 1, v_3, v_4 \Leftrightarrow 1, v_5)$. À partir de ce point nous appliquons les équations $X_3 \mathrel{\hat{=}} t_2 \cdot X_3$ et $X_3 \mathrel{\hat{=}} t_4 \cdot X_3$ et simultanément les transitions $p \overset{t_2}{\to} q$ et $q \overset{t_4}{\to} p$ et nous obtenons $(v_1, v_2, v_3, v_4, v_5) \in [L]$.
     - $2^{v_4 - 1} < v_3 + 1 \leq 2^{v_4}$
       Fixons un b avec $v_3 + 1 = 2^{v_4 - 1} + b$ et $0 < b \leq 2^{v_4 - 1}$. Considérons le vecteur $(v_3, v_4 \Leftrightarrow 1, 2^{v_4 - 1} \Leftrightarrow 1, v_4 \Leftrightarrow 1, v_5)$. Ce vecteur satisfait R et par hypothèse d'induction il existe un calcul $X_1 \overset{init}{\to} X_2 \| X_3 \to^* X_2^k \| X_4^j \| X_3$ (on peut montrer $k = v_3 \Leftrightarrow 2b \geq 0$ et $j = b$) et un calcul $p \overset{init}{\to} p \to^* p$ tel que le deux produisent une séquence σ avec $[\sigma] = (v_1, v_2 \Leftrightarrow 1, v_3, v_4 \Leftrightarrow 1, v_5)$. À partir de ce point nous appliquons les équations et nous pouvons produire avec le BPP et l'automate simultanément la séquence $t_2 t_3^b t_4 t_1^{v_1 - v_3}$. Cela donne une séquence σ' avec $[\sigma'] = (v_1, v_4, 2^{v_4 - 1} \Leftrightarrow 1 + b, v_4, v_5)$. Parce que $v_3 = 2^{v_4 - 1} \Leftrightarrow 1 + b$, $(v_1, v_2, v_3, v_4, v_5) \in [L]$.

2. $a_2 = a_4 + 1$:
   Un argument similaire que dans le cas $a2 = a4$.

□

Le corollaire suivant est une conséquence directe de la Proposition 2.2. Ce corollaire suit aussi directement du fait que l'ensemble des marquages atteignables d'un réseau de Petri peut être non-semilinéaire [HP79].

Corollaire 2.3 :

Les ω-langages de réseaux de Petri ne sont pas semilinéaires.

2.8 L'expressivité

Dans cette section nous étudions l'expressivité des différentes classes de systèmes que nous considérons dans cette thèse. Nous étudions d'abord les différentes classes par rapport aux langages *finis* qu'ils génèrent. Ensuite, nous considérons l'expressivité par rapport aux ω-langages.

2.8.1 Langages de mots finis

Il est clair que BPA et BPP sont plus expressifs que RP. Les processus BPA génèrent les langages hors-contexte [BBK87] (donc les mêmes que les automates à pile (PDA)) et Christensen a montre dans [Chr93] que BPP n'est pas comparable avec BPA. Par exemple, le langage $\{a^n b^n \ : \ n > 0\}$ peut être défini par le BPA avec une équation

$$X_1 \mathrel{\widehat{=}} a \cdot X \cdot b + a \cdot b$$

Ce langage ne peut pas être généré par un BPP. Par contre le langage $\{\sigma \in \Sigma^+ \ : \ |\sigma|_a = |\sigma|_b = |\sigma|_c\}$, qui n'est pas hors-contexte peut être généré par le BPP:

$$X \mathrel{\widehat{=}} a \cdot (b||c||X + b||c) + b \cdot (a||c||X + a||c) + c \cdot (a||b||X + a||b)$$

PA est par conséquent strictement plus expressif que BPA et BPP. Nous pouvons montrer que PA est même strictement plus expressive que l'union de BPA et BPP (BPA $\uplus$ BPP), i.e. PA peut généré des langages qui ne sont ni généré par BPP ni par BPA, par exemple

$$\{\sigma \in \Sigma^+ \ : \ \sigma|_{\{a,b\}} \in \{a^n b^n \ : \ n > 0\} \text{ et } \sigma|_{\{c,d\}} \in \{c^n d^n \ : \ n > 0\}\}$$

Ce langage peut être généré par un PA qui est la composition parallèle de deux processus BPA. Cet exemple montre aussi que BPA n'est pas fermé par composition parallèle.

tel-00004890, version 1 - 19 Feb 2004

Christensen a montré que BPP n'est pas fermée par composition séquentielle. Nous écrivons $\|^k$-BPA (resp. $\odot^k$-BPP) pour la classe de processus obtenue par composition parallèle (resp. séquentielle) de k processus BPA (resp. BPP). Soit $\|$-BPA $= \bigcup k \in I\!N \|^k$-BPA et $\odot$-BPP$= \bigcup k \in I\!N \odot$-BPA.

Lemme 2.10 montre que les processus BPP peuvent être définie comme un Réseau de Petri et puisque les réseaux de Petri sont fermé par concaténation, $\odot$-BPP est inclus dans BPA. Par contre il y a un réseau de Petri qui définit le langage $\{a^n b^n \ : \ n > 0\}$, qui n'est pas définissable par un $\odot$-BPP. Finalement, les réseaux de Petri ne sont pas comparables avec BPA [Chr93]. Nous obtenons l'image 2.2 ou les arêtes représentent l'inclusion stricte entre les classes de langages.

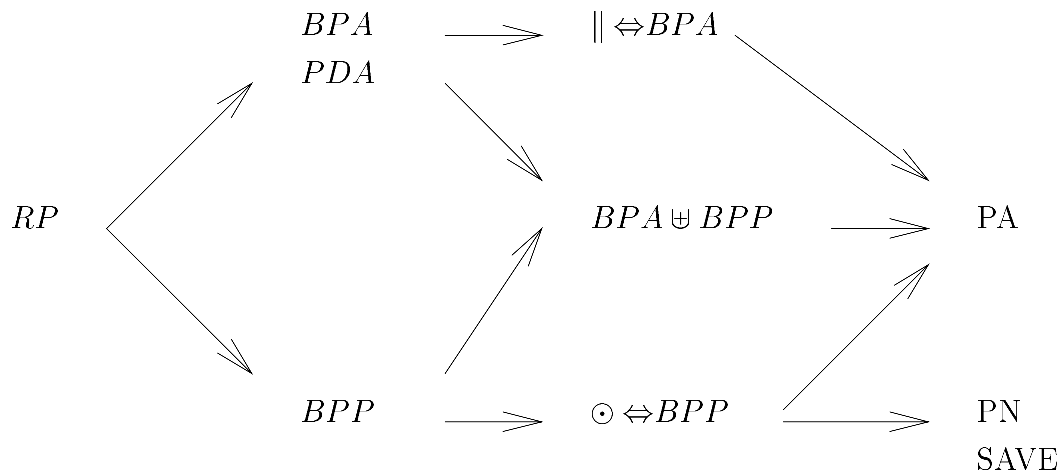


Fig. 2.2 – *Les systèmes*

2.8.2 ω-langages

La situation dans la figure 2.2 ne change évidemment pas si nous considérons les ω-langages générés par les différentes classes de processus. Les ω-langages simples sont incomparables avec tous les ω-langages de RP,BPA,BPP,PN,PA. Les classes des ω-langages simples, sans hauteurs d'étoile, ω-régulier et ω-hors-contexte forment une hiérarchie stricte.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Chapitre 3

La complexité du μ-calcul linéaire pour les réseaux de Petri

Dans ce chapitre nous étudions la complexité du problème du *model-checking* des réseaux de Petri par rapport au μ-calcul propositionnel linéaire (en considérant les transitions comme propositions atomiques). Esparza a montré que ce problème est décidable [Esp94]. La complexité en espace de son algorithme est exponentielle dans la taille du système (nombre de places du réseau) et double-exponentielle dans la taille de la formule. Nous montrons dans ce chapitre que la complexité dans la taille de la formule est polynômiale. Nous montrons que nous ne pouvons essentiellement pas faire mieux en prouvant que notre borne supérieure pour la complexité est presque optimale. Pour cela nous prouvons que le problème est PSPACE-difficile dans la taille de la formule et EXPSPACE-difficile dans la taille du système. Nous montrons que cette borne inférieure peut déjà être établie pour la sous-classe BPP. Cela est assez surprenant puisque BPP est une sous-classe simple des réseaux de Petri (chaque transition a au plus une entrée). Nous montrons aussi que les résultats restent les mêmes en passant à LTL, une logique moins expressive que le μ-calcul propositionnel linéaire.

Pour obtenir la borne supérieure de la complexité en espace nous utilisons les systèmes d'addition de vecteurs avec états (SAVE), définis dans le chapitre 2. Nous réduisons le problème du *model-checking* au *problème de répétition d'un état* dans les SAVE, c.-à-.d. le problème d'existence d'une séquence de transitions qui passe infiniment souvent par un état de contrôle donné. Nous analysons ce problème dans les SAVE en utilisant les techniques de [Rac78] et [RY86] qui prouvent des bornes supérieures pour la complexité en espace du problème du caractère borné des réseaux de Petri et des SAVE. Ces techniques consistent essentiellement à montrer que, s'il y a une séquence de transitions dans un SAVE qui montre que ce SAVE est non-borné, alors il y a une telle séquence d'une longueur qui peut être bornée par une constante qui ne dépend que de la taille du système.

La borne inférieure est obtenue par réduction d'un problème EXPSPACE-difficile dans les réseaux de Petri.

Les résultats de ce chapitre ont été publiés dans [Hab97].

tel-00004890, version 1 - 19 Feb 2004

3.1 Préliminaires

Dans cette section nous donnons quelques définitions et résultats préliminaires sur les ω-automates, le μ-calcul linéaire et les classes de complexité que nous considérons.

3.1.1 Résultats de complexité sur les ω-automates

Dans la section 3.1.2 nous construisons "à la volée" l'automate de Büchi $\mathcal{A}_\varphi$ qui est équivalent à une formule φ du μ-calcul linéaire pour résoudre le problème de model-checking. Cette construction comprend la complémentation et l'intersection d'automates de Büchi. Pour la preuve d'une borne inférieure de la complexité en espace du problème de model-checking dans la section 3.3 nous devons construire l'automate $\mathcal{A}_\varphi$ "à la volée", c.-à-d. nous construisons l'information sur l'automate seulement quand elle est utilisée dans notre algorithme de model-checking. Nous avons donc besoin de bornes sur la taille d'une description d'un état et la taille de l'automate dans ces constructions. Les lemmes suivants donnent ses bornes. Nous commençons avec l'intersection. Ensuite, pour la complémentation nous avons besoin de la déterminisation.

Lemme 3.1 :

> Pour deux automates de Büchi $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) avec n_1 (resp. n_2) états, où la description d'un état a la taille d_1 (resp. d_2), il existe un automate de Büchi $\mathcal{A}_3$ et qui a $2n_1n_2$ états tel que $L(\mathcal{A}_3) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. Pendant la construction de $\mathcal{A}_3$ la description d'un état a la taille $d_1 + d_2 + 1$.

Preuve:
Ce lemme se déduit de la construction habituelle pour l'intersection de deux automates de Büchi (voir par exemple [Var96]). □

Lemme 3.2 :

> Pour chaque automate de Büchi $\mathcal{A}_1$ avec n états il existe un automate de Rabin déterministe $\mathcal{A}_2$ avec $L(\mathcal{A}_1) = L(\mathcal{A}_2)$ et qui a $2^{O(nlog(n))}$ états et $O(n)$ couples accepteurs. Pendant la construction de $\mathcal{A}_2$ une description d'un état a la taille $O(n^2)$.

Preuve:
Ce lemme se déduit de l'algorithme de déterminisation de Safra [Saf88]. La taille d'une description d'un état de $\mathcal{A}_2$ est $O(n^2)$ parce qu'un état est un arbre avec au maximum n nœuds et chaque nœud est constitué d'un sous-ensemble de l'ensemble d'états de $\mathcal{A}_1$. Chaque couple accepteur est déterminé de manière unique par un nœud de l'arbre. □

Lemme 3.3 :

Pour chaque automate de Streett $\mathcal{A}_1$ avec n états (avec une description d'un état de taille d) et h couples accepteurs il existe un automate de Büchi $\mathcal{A}_2$ tel que $L(\mathcal{A}_1) = L(\mathcal{A}_2)$ et qui a $n \cdot 2^{O(h)}$ états. Pendant la construction de $\mathcal{A}_2$ la description d'un état a la taille $d \cdot O(h^2)$.

Preuve:
Ce déduit facilement de [Saf88]. □

Lemme 3.4 :

Pour chaque automate de Büchi $\mathcal{A}_1$ avec n états il existe un automate de Büchi $\mathcal{A}_2$ tel que $L(\mathcal{A}_2) = \overline{L(\mathcal{A}_1)}$ qui a $2^{O(nlog(n))}$ états. Pendant la construction de $\mathcal{A}_2$ la description d'un état a la taille $O(n^4)$.

Preuve:
Nous commençons par déterminiser $\mathcal{A}_1$ en utilisant le Lemme 3.2. Soit $\mathcal{R}$ l'automate de Rabin ainsi obtenu. Ensuite nous construisons un automate $\mathcal{S}$ en complétant $\mathcal{R}$ et en l'interprétant comme automate de Streett. Finalement, nous obtenons un automate de Büchi de la taille requise d'après le Lemme 3.3. □

3.1.2 Le μ-calcul propositionnel linéaire

Dans cette section nous définissons le μ-calcul propositionnel linéaire [Var88] et nous donnons des résultats concernant sa relation avec les automates de Büchi.

Définition 3.1 (Formule du μ-calcul linéaire) :

Soit $\mathcal{X}$ un ensemble fini de *variables*. Alors l'ensemble des *formules du μ-calcul propositionnel linéaire* est donné par la grammaire suivante:

$$\varphi ::= \; Z \in \mathcal{X} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \mu Z.\varphi(Z)$$

avec une condition de monotonie: Étant donnée une formule $\mu Z.\varphi(Z)$, chaque occurrence libre de Z dans $\varphi(Z)$ doit être dans la portée d'un nombre pair de négations.

tel-00004890, version 1 - 19 Feb 2004

μ est l'opérateur de plus petit point fixe. Nous utilisons l'abréviation $\nu Z.\varphi(Z) = \neg\mu Z.\neg\varphi(\neg Z)$. ν est l'opérateur de plus grand point fixe.

Définition 3.2 (Modèle) :

Un *modèle* $\mathcal{M}$ d'une formule associe à chaque variable $Z \in \mathcal{X}$ un sous-ensemble $\mathcal{M}(Z) \subseteq I\!N$. Les modèles sont généralisés à toutes les formules par:

$$\begin{array}{rcl} \mathcal{M}(\neg\varphi) & = & I\!N \setminus \mathcal{M}(\varphi) \\ \mathcal{M}(\varphi \wedge \psi) & = & \mathcal{M}(\varphi) \cap \mathcal{M}(\psi) \\ \mathcal{M}(\bigcirc\varphi) & = & \{i \mid i+1 \in \mathcal{M}(\varphi)\} \\ \mathcal{M}(\mu Z.\varphi(Z)) & = & \bigcap\{A \subseteq I\!N \mid \mathcal{M}[Z \mapsto A](\varphi) \subseteq A\} \end{array}$$

où $\mathcal{M}[Z \mapsto A]$ est le modèle qui associe A à Z et coïncide avec $\mathcal{M}$ sur les autres variables.

Définition 3.3 (Interprétation) :

Soit $\mathcal{Y} \subseteq \mathcal{X}$ l'ensemble des variables propositionnelles libres dans une formule φ et $\Sigma = 2^{\mathcal{Y}}$. Alors, un modèle $\mathcal{M}$ de φ détermine une séquence infinie $\sigma_{\mathcal{M}} \in \Sigma^{\omega}$ donnée par $\sigma_{\mathcal{M}}(i) = \{Z \in \Sigma \mid i \in \mathcal{M}(Z)\}$. L'*interprétation* de φ (notée $[\![\varphi]\!]$) est définie par

$$[\![\varphi]\!] = \{\sigma_{\mathcal{M}} \mid 0 \in \mathcal{M}(\varphi) \text{ pour un modèle } \mathcal{M}\}$$

i.e. l'ensemble de toutes les séquences infinies qui satisfont φ.

Théorème 3.1 :

Soit φ une formule du μ-calcul linéaire φ. Alors, il existe un automate de Büchi $\mathcal{A}_{\varphi}$, tel que $L(\mathcal{A}_{\varphi}) = [\![\varphi]\!]$ et vice versa.

Une traduction des automates de Büchi vers le μ-calcul linéaire est par exemple donnée dans [Par81] ou [Dam92]. La construction d'un automate à partir d'une formule est détaillée dans [Var88]. Ici, nous donnons les grandes lignes de la construction, dont nous avons besoin pour prouver une borne supérieure de la complexité du problème de model-checking.

Soit n la taille (nombre de sous-formules) de φ. Pour construire l'automate de Büchi correspondant nous transformons d'abord φ dans une formule φ' en forme normale positive, c.-à-d. que toutes les sous-formules qui ne sont pas des variables propositionnelles sont positives. Cela peut être fait avec une augmentation linéaire de la taille de la formule en utilisant des règles de transformation standards comme $\neg\mu Z.\varphi(Z) = \nu Z.\neg\varphi(\neg Z)$.

Après, $[\![\varphi']\!]$ est donnée par la projection sur les variables propositionnelles libres du produit de deux automates de Büchi $\mathcal{A}_1$ et $\mathcal{A}_2$ définis comme suit:

$\mathcal{A}_1$ correspond à la formule sans tenir compte des plus petits points fixes, c.-à-d. il accepte les *pre-modèles* de la formule. $\mathcal{A}_1$ accepte des séquences qui ne sont pas des modèles, parce qu'il n'exige pas que l'évaluation des plus petits points fixes termine.

tel-00004890, version 1 - 19 Feb 2004

Pour obtenir les modèles, $\mathcal{A}_2$ est le complément d'un automate de Büchi $\mathcal{A}_3$ qui accepte toutes les séquences où un plus petit point fixe est régénéré infiniment souvent (voir [Var88] pour les détails de la construction). L'automate $\mathcal{A}_3$ a $O(n^2)$ états. En utilisant le Lemme 3.4 nous pouvons construire $\mathcal{A}_2$ avec $2^{O(n^2 log(n))}$ états et une description d'un état pendant la construction de $\mathcal{A}_2$ a une taille polynômiale. Nous obtenons donc d'après le Lemme 3.1:

Lemme 3.5 :

Soit φ une formule du μ-calcul linéaire de taille n. Alors, il existe un automate de Büchi $\mathcal{A}_\varphi$ avec $2^{O(n^2 log(n))}$ états tel que $L(\mathcal{A}_\varphi) = [\![\varphi]\!]$. En plus, pendant la construction de $\mathcal{A}_\varphi$, la description d'un état a une taille polynômiale en n.

3.1.3 Les classes de complexité

Nous utilisons les classes de complexité habituelles PSPACE et EXPSPACE définies par exemple dans [Pap94]. Un problème de décision est en $SPACE(f(n))$ s'il y a une machine de Turing déterministe qui le décide et qui est bornée en espace par $f(n)$. Nous avons

$$PSPACE = \bigcup_{k \in I\!N} SPACE(n^k)$$

et

$$EXPSPACE = \bigcup_{k \in I\!N} SPACE(2^{n^k}).$$

Soit

$$ESPACE = \bigcup_{k \in I\!N} SPACE(k^n)$$

Nous modifions le Lemme 20.1 de [Pap94] qui concerne la complexité en temps pour la complexité en espace et nous obtenons:

Lemme 3.6 :

Pour chaque problème P dans EXPSPACE il y a un problème P' dans ESPACE tel que P se réduit (en temps polynômial) à P'.

Avec ce lemme nous montrons facilement:

Lemme 3.7 :

Un problème est EXPSPACE-difficile ssi il est ESPACE-difficile.

et de la même manière

Lemme 3.8 :

Si un problème est $SPACE(2^n)$-difficile il est EXPSPACE-difficile.

Pour montrer qu'un problème est EXPSPACE-difficile il suffit donc de montrer qu'il est $SPACE(2^n)$-difficile.

tel-00004890, version 1 - 19 Feb 2004

3.2 Résultat de complexité pour les SAVE

Le problème du model-checking pour les réseaux de Petri et les processus BPP par rapport au μ-calcul linéaire est résolu en le transformant vers le *problème de répétition d'un état de contrôle* d'un SAVE. Le problème de répétition d'un état de contrôle d'un SAVE est le problème d'existence d'une séquence de transitions commençant avec la configuration initiale et qui visite infiniment souvent un état de contrôle donné. Dans ce chapitre nous analysons en détail la complexité en espace de ce problème. Nous donnons une borne supérieure qui est paramétrée par la dimension du SAVE, le nombre d'états et la valeur absolue la plus grande des composantes des vecteurs de l'ensemble d'addition. La technique de preuve que nous utilisons est proche des techniques utilisées dans [Rac78] et [RY86] qui établissent des bornes supérieures pour le problème du caractère non-borné des réseaux de Petri.

3.2.1 Notation

Pour prouver la borne supérieure nous avons besoin d'introduire quelques définitions et notations. Toutes ces définitions et notations peuvent être essentiellement trouvées dans [Rac78] et [RY86].

Définition 3.4 (Chemin et effet) :

Soit $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ un SAVE de dimension k. Un *chemin* de longueur m à partir de $(q, \vec{v})$ est une séquence de paires $(q_1, \vec{w}_1)$,$(q_2, \vec{w}_2)$, ... , $(q_m, \vec{w}_m)$ telle que

- $(q_1, \vec{w}_1) = (q, \vec{v})$,
- $\forall i$ avec $1 \leq i \leq m$. $q_i \in Q$ et $\vec{w}_i \in \mathbb{Z}^k$,
- $\forall i$ avec $1 \leq i < m$. $q_i \rightarrow (q_{i+1}, \vec{w}_{i+1} \Leftrightarrow \vec{w}_i) \in \delta$.

L'*effet* d'un chemin est le vecteur $\vec{w}_m \Leftrightarrow \vec{w}_1$.

Intuitivement, un chemin est une séquence de transitions qui *ne* respecte *pas* la propriété: *chaque vecteur qui apparaît dans la séquence ne contient pas des composantes négatives.*

Définition 3.5 (vecteurs et chemins bornés) :

Soit $\vec{w} \in \mathbb{Z}^k$ et $0 \leq i \leq k$. Le vecteur $\vec{w}$ est appelé *i borné* ssi $\forall j$ avec $1 \leq j \leq i$. $\vec{w}(j) \geq 0$. Pour un $r \in \mathbb{N}^+$ tel que $0 \leq \vec{w}(j) < r$ pour $1 \leq j \leq i$, $\vec{w}$ est appelé *i-r borné*. Un chemin d'un SAVE est appelé *i borné* (resp. *i-r borné*) si tous les vecteurs contenus dans le chemin sont i bornés (resp. i-r bornés).

tel-00004890, version 1 - 19 Feb 2004

Remarque: Pour un chemin k borné $(q_1, \vec{w}_1), (q_2, \vec{w}_2), \ldots, (q_m, \vec{w}_m)$ d'un SAVE à k dimension à partir de la configuration initiale nous avons $(q_1, \vec{w}_1) \rightsquigarrow (q_2, \vec{w}_2) \rightsquigarrow \ldots \rightsquigarrow (q_m, \vec{w}_m)$. Un chemin k borné correspond donc à une séquence de transitions.

Définition 3.6 (i boucle) :

> Une i boucle est un chemin $(q_1, \vec{w}_1), \ldots, (q_m, \vec{w}_m)$ où $q_1 = q_m$ et $\vec{w}_1(j) = \vec{w}_m(j)$ pour tout j avec $1 \leq j \leq i$.

Nous pouvons donner maintenant notre résultat principal sur les SAVE.

3.2.2 Problème de répétition d'un état de contrôle

Dans cette partie nous obtenons une borne supérieure de la complexité en espace du problème suivant:

Problème 3.1 :

> Soit $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ un SAVE. Est-ce qu'il existe une séquence de transition à partir de la configuration initiale qui visite infiniment souvent un état de contrôle $q \in Q$ donné?

Du fait que $\leq$ sur $\mathbb{N}^k$ est un bel ordre (il n'y a pas de séquences infinis $\vec{v}_1, \vec{v}_2, \ldots$ avec $\vec{v}_i \not\leq \vec{v}_j$ pour tout $i < j$), le problème 3.1 est équivalent au problème:

Problème 3.2 :

> Soit $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ un SAVE et $q \in Q$ un état de contrôle. Est-ce qu'ils existent des configurations $(q, \vec{w})$, $(q, \vec{w}')$ telles que $(q_0, \vec{v}_0) \rightsquigarrow^* (q, \vec{w}) \rightsquigarrow^+ (q, \vec{w}')$ et $\vec{w}' \geq \vec{w}$?

Ce problème est très proche du problème de couverture et du caractère non-borné d'un réseau de Petri (voir [Rac78]). En effet, si nous considérons un SAVE avec un état et si nous remplaçons la condition $\vec{w}' \geq \vec{w}$ par $\vec{w}' > \vec{w}$, nous obtenons une condition nécessaire et suffisante pour le caractère non-borné d'un réseau de Petri. Ainsi, pour obtenir une borne supérieure de notre problème nous adaptons la preuve d'une borne supérieure de la complexité du problème de couverture [Rac78]. Nous avons besoin d'une borne supérieure paramétrée par le nombre d'états du SAVE. Nous suivons donc l'approche de Rosier et Yen [RY86] qui analysent en détail le problème du caractère non-borné pour les réseaux de Petri et les SAVE.

Nous nous intéressons à la complexité en espace du problème 3.2. Par conséquent, nous montrons que s'il y a une séquence de transitions qui satisfait la propriété du problème 3.2, alors il y a une séquence de transitions d'une certaine longueur qui la satisfait. Alors, il suffit de tester si dans toutes les séquences jusqu'à cette longueur il y en a une qui satisfait la propriété. Nous montrons qu'il y a une borne supérieure sur la longueur par induction sur la dimension du SAVE. Nous montrons qu'il y a une borne supérieure pour les chemins

tel-00004890, version 1 - 19 Feb 2004

qui sont "corrects" (dans le sens de $\rightsquigarrow$) pour les premières i dimensions, c.-à-d. qui sont i bornés. Finalement, nous obtenons une borne pour les chemins k bornés qui correspondent à des séquences de transitions.

Pour commencer, considérons un SAVE $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ à k dimensions et n états. Nous supposons que les valeurs absolues des composantes des vecteurs de A sont inférieures à l. Soit $q \in Q$.

Définition 3.7 (chemin qui répète q) :

> Un *chemin qui répète* q à partir de $(q', \vec{v})$ dans un SAVE est un chemin à partir de $(q', \vec{v})$ de la forme $(q_1, \vec{w}_1), (q_2, \vec{w}_2), \ldots, (q_m, \vec{w}_m)$ tel que $\exists j.\ 1 \leq j < m$ avec $q_j = q$, $q_m = q$ et $\vec{w}_m \geq \vec{w}_j$.

Un chemin k borné qui répète q est une séquence de transition qui satisfait la propriété du Problème 3.2.

Nous avons besoin de la définition suivante dans les preuves de cette section.

Définition 3.8 (Élimination d'une i-boucle dans un chemin) :

> Soit $\rho = (q_1, \vec{w}_1), (q_2, \vec{w}_2), \ldots, (q_m, \vec{w}_m)$ un chemin avec une i boucle de la forme $(q_{j_1}, \vec{w}_{j_1}), \ldots, (q_{j_2}, \vec{w}_{j_2})$ avec $1 \leq j_1 < j_2 \leq m$. Alors, le chemin σ' où la boucle est éliminée est donné par
>
> $$\rho' = (q_1, \vec{w}_1), \ldots, (q_{j_1}, \vec{w}_{j_1}), (q_{j_2+1}, \vec{w}'_{j_2+1}), \ldots, (q_m, \vec{w}'_m)$$
>
> où pour j avec $j_2 + 1 \leq j \leq m$, nous avons $\vec{w}'_j = \vec{w}_{i_1} \Leftrightarrow \vec{w}_{i_2} + \vec{w}_j$.

Remarque: Dans un chemin σ avec une i boucle, les états et les i premières composantes des vecteurs qui ne font pas partie de la boucle ne changent pas en éliminant la boucle.

Définition 3.9 (Fonction m) :

> Soit i avec $0 \leq i \leq k$. Pour chaque paire $(q', \vec{v})$ avec $q' \in Q$ et $\vec{v} \in \mathbb{Z}^k$, $m(i, q', \vec{v})$ est la longueur du plus petit chemin i borné à partir de $(q', \vec{v})$ qui répète q, si un chemin i borné existe, sinon $m(i, q', \vec{v})$ est égale à 0.

Définition 3.10 (Fonction f) :

> Soit $f(i) = max\{m(i, q', \vec{v}) \mid q' \in Q, \vec{v} \in \mathbb{Z}^k\}$.

Remarque: Cette définition ne dépend pas de l'état initial et du vecteur de début du SAVE.

Nous montrons une borne supérieure sur $f(i)$ par induction. Au bout du compte, nous sommes seulement intéressés par $f(k)$.

Pour la preuve du pas d'induction nous avons besoin d'une borne supérieure sur la longueur des chemins i-r bornés qui répètent q dans le SAVE $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$. Pour montrer cette borne nous utilisons un lemme de la programmation linéaire, qui montre

tel-00004890, version 1 - 19 Feb 2004

pour un certain type de système d'équations que s'il y a une solution de ce système, il y a une solution où les valeurs des variables sont bornées. Ce lemme est de [Rac78] et la preuve se trouve dans [BT76].

Lemme 3.9 :

> Soient $d_1, d_2 \in \mathbb{N}^+$ et B une $d_1 \times d_2$ matrice d'entiers. Soient b une $d_1 \times 1$ matrice d'entiers et $d \geq d_2$ une borne supérieure sur les valeurs absolues des entiers de B et b. S'il existe un vecteur $\vec{v} \in \mathbb{N}^{d_2}$ qui est une solution de $B\vec{v} \geq b$, alors pour une constante c indépendante de d, d_1, d_2, il existe un vecteur $\vec{v} \in \mathbb{N}^{d_2}$ tel que $B\vec{v} \geq b$ et $\vec{v}(i) < d^{cd_1}$ pour tout $i, 1 \leq i \leq d_2$.

Lemme 3.10 :

> S'il existe un chemin i-r borné à partir de $(q', \vec{v})$ qui répète q, alors il existe un chemin i-r borné à partir de $(q', \vec{v})$ qui répète q avec une taille $< (rnl)^{k^c}$, où c est une constante indépendante de r, l, k et n.

Preuve:
La preuve est proche des preuves similaires de [Rac78] et [RY86]. Nous appelons ρ le chemin i-r borné à partir de $(q', \vec{v})$ qui répète q avec $\rho = (q_1, \vec{w}_1), \ldots, (q_m, \vec{w}_m)$. Puisque ρ répète q nous pouvons obtenir deux chemins ρ_1 et ρ_2 tels que $\rho_1 = (q_1, \vec{w}_1), \ldots, (q_{m_0}, \vec{w}_{m_0})$ avec $q_{m_0} = q$ et $\rho_2 = (q_{m_0}, \vec{w}_{m_0}) \ldots (q_m, \vec{w}_m)$ où $q_m = q$ et $\vec{w}_m \geq \vec{w}_{m_0}$. Nous pouvons supposer que ρ_1 n'est pas plus long que $r^i n \leq r^k n$, parce que sinon il y aurait deux paires $(q_{j_1}, \vec{w}_{j_1})$ et $(q_{j_2}, \vec{w}_{j_2})$ avec $1 \leq j_1 < j_2 \leq m_0$ telles que $q_{j_1} = q_{j_2}$, et $\vec{w}_{j_1}$ et $\vec{w}_{j_2}$ s'accordent sur les premières i composantes. Dans ce cas nous pourrions obtenir un nouveau chemin plus court qui serait i-r borné et qui répéterait q en éliminant de ρ_1 la i boucle entre $(q_{j_1}, \vec{w}_{j_1})$ et $(q_{j_2}, \vec{w}_{j_2})$. Cela ne changerait pas la propriété $\vec{w}_m \geq \vec{w}_{m_0}$.

Pour obtenir une borne supérieure sur la longueur de ρ_2 nous montrons essentiellement comment éliminer les i boucles. Nous ne pouvons pas appliquer la même technique utilisée pour ρ_1, parce que nous devons assurer que la propriété $\vec{w}_m \geq \vec{w}_{m_0}$ est toujours vraie après élimination d'une boucle.

Le chemin ρ_2 peut être décomposé en plusieurs i boucles et le reste du chemin, appelé ρ_3, tel qu'aucune paire $(q_j, \vec{w}_j)$ faisant partie d'une i boucle, n'apparaît en ρ_3. Cela garantit que chaque boucle peut toujours être exécutée après élimination des autres boucles. La taille de ρ_3 est strictement inférieure à $(r^k n + 1)^2$. Si elle était plus grande, nous pourrions toujours trouver une autre i boucle telle qu'aucune paire dans la boucle n'apparaît en dehors de la boucle.

La taille d'une i boucle peut être choisie inférieure ou égale à $r^i n \leq r^k n$. Les effets des boucles sont donc la somme d'au plus $r^k n$ vecteurs qui ont des valeurs absolues d'au plus l. Par conséquent, il y a au plus $(2 \cdot l r^k n + 1)^k$ d'effets de boucles différents. Appelons les effets de boucles différents l_j.

Du fait que ρ_2 est un chemin qui répète q, nous savons que le système d'équation $n_1 l_1 + \cdots + n_p l_p + e(\rho_3) \geq \vec{0}$, où $e(\rho_3)$ est l'effet de ρ_3, a une solution dans $\mathbb{N}^p$.

tel-00004890, version 1 - 19 Feb 2004

En posant $d = max(l(r^k n+1)^2, (2 \cdot lr^k n+1)^k)$ et $d_1 = k$ et en utilisant le Lemme 3.9 nous voyons qu'il y a une solution avec pour tous les j, $n_j < (lrn)^{k^c}$ pour une certaine constante c. Cela nous donne une borne sur le nombre de fois une certaine boucle doit être exécutée. Grâce à la construction de ρ_3 chaque boucle avec $n_j > 0$ peut être exécutée à partir d'une paire de ρ_3. Par conséquent nous pouvons construire à partir de ρ_3 un chemin i-r borné qui répète q avec une taille $< (lrn)^{k^c} r^k n + (r^k n+1)^2 + r^k n < (lrn)^{k^{c'}}$. □

Finalement, nous sommes en mesure d'établir par induction une borne supérieure pour $f(i)$.

Lemme 3.11 :

$f(0) < (l^2 n)^{k^c}$ pour une constante c indépendante de l, k et n.

Preuve:
Ce lemme se déduit du Lemme 3.10 et du fait qu'un chemin 0 borné qui répète q est trivialement un chemin 0-1 borné qui répète q. □

Lemme 3.12 :

$\forall i \geq 0.\ f(i+1) < (l^2 n f(i))^{k^c}$ pour une constante c indépendante de l, k et n.

Preuve:
Considérons un chemin quelconque i+1 borné à partir de $(q', \vec{v})$ qui répète q. Ce chemin est donné par $\rho = (q_1, \vec{w}_1) \ldots (q_m, \vec{w}_m)$. Nous considérons deux cas:

- Cas 1: Si ρ est $(i+1)$-$lf(i)$ bornée, alors avec lemme 3.10 il existe un chemin i+1 borné qui répète q avec taille $< (l^2 n f(i))^{k^c}$.
- Cas 2: ρ n'est pas $(i+1)$-$lf(i)$ borné. D'abord, nous pouvons construire facilement un chemin $\rho' = \rho\rho_1$ tel que $(q_m, \vec{w}_m)\rho_1$ est $(i+1)$ borné et répète q. ρ n'est pas $(i+1)$-$lf(i)$ borné. Donc, il existe un premier point m_0 sur ce chemin, où le vecteur n'est pas $lf(i)$ borné. Sans perte de généralité nous supposons que $\vec{w}_{m_0}(i+1) \geq lf(i)$ et $(q_{m_0}, \vec{w}_{m_0}) \ldots (q_m, \vec{w}_m)\rho_1$ (appelé ρ_2) est un chemin $(i+1)$ borné à partir de $(q_{m_0}, \vec{w}_{m_0})$ qui répète q.

  Par ailleurs, nous pouvons supposer que $m_0 < (lf(i))^{i+1} n$, parce que sinon il y aurait deux paires $(q_{j_1}, \vec{w}_{j_1})$ et $(q_{j_2}, \vec{w}_{j_2})$ avec $1 \leq j_1 < j_2 \leq m_0$ tel que $q_{j_1} = q_{j_2}$ et $\vec{w}_{j_1}$ et $\vec{w}_{j_2}$ s'accordent sur les premières i+1 composantes. Et nous pourrions obtenir un nouveau chemin i+1 borné qui répète q en éliminant de ρ la boucle entre $(q_{j_1}, \vec{w}_{j_1})$ et $(q_{j_2}, \vec{w}_{j_2})$.

  Parce que ρ_2 est un chemin $(i+1)$ borné qui répète q, il est trivialement un chemin i borné qui répète q. Donc, en utilisant l'hypothèse d'induction il y a un chemin i

tel-00004890, version 1 - 19 Feb 2004

borné à partir de $(q_{m_0}, \vec{w}_{m_0})$ qui répète q avec une taille $\leq f(i)$. A cause du fait que $\vec{w}_{m_0}(i+1) \geq lf(i)$ et chaque transition enlève au plus l de chaque composante ce chemin est aussi $i+1$ borné.

En mettant ensemble les deux chemins nous obtenons un chemin i+1 borné qui répète q de taille $< (lf(i))^{i+1}n + f(i) < (l^2 nf(i))^{k^c}$.

□

D'après les Lemmes 3.11 et 3.12 nous avons $f(k) < (ln)^{(k^c)^k} = (ln)^{2^{cklog(k)}}$ pour une constante c. Un algorithme non-déterministe pour résoudre le Problème 3.2 devine un chemin à partir de la configuration initiale qui répète q. La longueur de ce chemin peut être bornée par $(ln)^{2^{cklog(k)}}$. Cet algorithme peut être exécuté avec un espace borné par $O((log(l) + log(n))2^{cklog(k)})$. D'où:

Théorème 3.2 :

> Étant donné un SAVE $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ à k dimension avec n états, où les valeurs absolues de vecteurs de A sont inférieurs à l, le Problème 3.2 peut être résolu en $O((log(l)+log(n))2^{cklog(k)})$ espace non-déterministe avec une constante c indépendante de l,k et n.

Nous utilisons ce théorème dans la partie suivante pour prouver une borne supérieure sur le problème de model-checking des réseaux de Petri par rapport au μ-calcul linéaire.

3.3 La borne supérieure

Dans cette partie nous donnons une borne supérieure pour la complexité en espace du problème de model-checking des réseaux de Petri par rapport au μ-calcul propositionnel linéaire. Ce problème a été montré décidable par Esparza [Esp94]. Sa définition de la sémantique d'une formule du μ-calcul propositionnel linéaire est différente de la notre. Il considère aussi des séquences finies pour exprimer des propriétés de blocages. Le problème de model-checking dans ce cas est réduit au problème d'atteignabilité dans les réseaux de Petri. Pour ce problème il n'existe jusqu'à présent pas de borne supérieure élémentaire. Une analyse détaillée paramétrée de la complexité du problème d'atteignabilité reste encore à faire. Cependant, ici nous ne considérons que le cas où une formule définit un ensemble de séquences infinies. Dans ce cas, l'algorithme de Esparza utilise un espace exponentiel dans la taille (nombre de places, dimension) du réseau de Petri et un espace double-exponentiel dans la taille de la formule. Cela vient du fait que dans l'algorithme d'Esparza l'automate de Büchi qui correspond à la formule est considéré comme un réseau de Petri, c.-à-d. les états de l'automate sont considérés comme des places dans un réseau de Petri. En ce qui nous concerne, nous utilisons pour représenter les états de l'automate les états d'un SAVE et nous pouvons par conséquent exploiter le fait que ce sont des états et non des places.

tel-00004890, version 1 - 19 Feb 2004

Cela nous permet de montrer que le problème du model-checking exige seulement un espace polynômial dans la taille de la formule, tout en demandant un espace exponentiel dans la taille du réseau de Petri.

Le problème du model-checking par rapport au μ-calcul est défini comme suit:

Définition 3.11 :

> Soit φ une formule interprétée sur Σ et $\mathcal{N}$ un réseau de Petri avec alphabet Σ.
> Le réseau de Petri *satisfait* la formule φ ssi $L(\mathcal{N}) \subseteq [\![\varphi]\!]$.

$L(\mathcal{N}) \subseteq [\![\varphi]\!]$ est équivalent à $L(\mathcal{N}) \cap [\![\neg\varphi]\!] = \emptyset$. Pour résoudre ce problème nous construisons un automate de Büchi $\mathcal{A}_{\neg\varphi}$, qui correspond à $[\![\neg\varphi]\!]$ et ensuite le "produit" entre cet automate et le réseau de Petri. Nous avons vu dans la section 2.6.2 qu'un réseau de Petri peut être vu comme un SAVE avec un seul état. Le produit que nous obtenons est donc un SAVE avec une condition d'acceptation "à la Büchi". Nous résolvons le problème du langage vide pour ce SAVE en utilisant les résultats de la section 3.2.

Définition 3.12 :

> Soit $\mathcal{S} = (\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ un SAVE avec un état et $\mathcal{A} = (\Sigma, Q', q'_0, \delta', F)$ un automate de Büchi. Le produit $\mathcal{S} \times \mathcal{A}$ est un SAVE $(\Sigma, \vec{v}_0, A, Q'', q''_0, \delta'')$ où
>
> - $Q'' = Q \times Q'$,
> - $q''_0 = (q_0, q'_0)$,
> - $((q_1, q'_1), b, (q_2, q'_2), \vec{a}) \in \delta''$ ssi $(q_1, b, q_2, \vec{a}) \in \delta$ et $(q'_1, b, q'_2) \in \delta'$.

Nous avons évidemment $L(\mathcal{S}) \cap L(\mathcal{A}) \subseteq L(\mathcal{S} \times \mathcal{A})$. Maintenant, pour obtenir l'autre direction de l'inclusion nous avons besoin de mettre une condition d'acceptation sur les chemins de $\mathcal{S} \times \mathcal{A}$. Le lemme suivant se déduit facilement.

Lemme 3.13 :

> Soit $\mathcal{S} = (\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ un SAVE avec un état et $\mathcal{A} = (Q', \Sigma, q'_0, \delta', F)$ un automate de Büchi. Alors $L(\mathcal{S}) \cap L(\mathcal{A}) \neq \emptyset$ si et seulement si il y a une séquence de transitions de $\mathcal{S} \times \mathcal{A} = (\Sigma, \vec{v}_0, A, Q'', q''_0, \delta'')$ à partir de l'état initial qui visite infiniment souvent un état $q'' = (q, q')$ où $q' \in F$.

En utilisant ce lemme, nous voyons que pour résoudre le problème du model-checking pour un réseau de Petri $\mathcal{N}$ qui est un SAVE $\mathcal{S}$ avec un état et une formule φ, nous devons résoudre quelques instances du problème 3.2 pour le SAVE $\mathcal{S} \times \mathcal{A}_{\neg\varphi}$.

Dans la section 3.2 nous avons donné une borne supérieure pour la complexité en espace du problème 3.1. Le problème est logarithmique dans le nombre d'états de contrôle du SAVE, parce que l'algorithme qui résout le problème d'une manière non-déterministe devine les chemins qui répètent q. Nous savons que la taille de l'automate de Büchi $\mathcal{A}_\varphi$ qui correspond à la formule du μ-calcul linéaire φ peut être exponentielle dans la taille de

tel-00004890, version 1 - 19 Feb 2004

la formule. Si nous devons construire tout cet automate, nous aurions besoin d'un espace exponentiel dans la taille de la formule. Mais, si nous pouvons construire l'automate "à la volée", nous obtenons un algorithme de model-checking, qui est polynômial dans la taille de la formule.

"À la volée" veut dire qu'étant donné φ, l'algorithme calcule l'information sur $\mathcal{A}_{\neg\varphi}$ seulement quand elle est utilisée dans l'algorithme pour résoudre le Problème 3.1. L'algorithme devine un état répété r et essaie de construire d'une manière non-déterministe une séquence de l'état initial vers r et une séquence de r vers r. Pour chaque pas de l'algorithme nous n'avons besoin de mémoriser que trois états de l'automate (l'état r, l'état courant et un possible successeur). Le Théorème 3.2 donne une borne supérieure sur le nombre de pas nécessaires. Ce nombre peut être mémorisée dans un espace polynômial dans la taille de la formule.

Pour montrer que nous pouvons construire $\mathcal{A}_{\neg\varphi}$ à la volée nous devons montrer (voir [VW94]) qu'une description d'un état de $\mathcal{A}_{\neg\varphi}$ a besoin au plus d'un espace polynômial et que toute l'information sur l'automate (Est-ce que deux états sont reliés par δ?, Est-ce qu'un état donné est répété?, etc.) peut être calculée en espace polynômial.

Nous prouvons cela par une inspection soigneuse de la construction de $\mathcal{A}_{\neg\varphi}$. Le lemme 3.5 montre qu'une description d'un état de $\mathcal{A}_{\neg\varphi}$ a une taille polynômiale. En plus, tous les pas de la construction (déterminisation, intersection) peuvent être effectués à la volée en espace polynômial.

En utilisant le théorème bien connu de Savitch [Sav70] nous pouvons éliminer le non-déterminisme dans le Théorème 3.2 et nous obtenons les deux théorèmes suivants. La taille d'un réseau de Petri est définie comme le nombre de places (dimensions).

Théorème 3.3 :

> Le problème de model-checking des réseaux de Petri par rapport au μ-calcul linéaire peut être résolu en PSPACE déterministe dans la taille de la formule.

Théorème 3.4 :

> Le problème de model-checking des réseaux de Petri par rapport au μ-calcul linéaire peut être résolu en espace $O(2^{cnlog(n)})$ déterministe pour une constante c où n est la taille du réseau de Petri.

Du fait que les processus BPP sont strictement moins expressifs que les réseaux de Petri nous obtenons les mêmes bornes supérieures pour les processus BPP. Dans le chapitre suivant nous allons montrer que nous ne pouvons essentiellement pas faire mieux.

3.4 La borne inférieure

Dans cette section nous montrons que le problème du model-checking de processus BPP par rapport à LTL est EXPSPACE-difficile dans la taille du système et PSPACE-difficile

tel-00004890, version 1 - 19 Feb 2004

dans la taille de la formule. La définition de LTL est donné dans la section 4.1.

Théorème 3.5 :

> Le problème du model-checking de processus BPP par rapport à LTL est PSPACE-difficile dans la taille de la formule.

Preuve:
Cela se déduit directement du fait [Var96] que le model-checking de système d'états finis est PSPACE-complet dans la taille de la formule. □

Nous utilisons le problème suivant pour obtenir une borne inférieure dans la taille du système.

Problème 3.3 :

> Soit $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ un SAVE à k dimensions tel que les vecteurs dans A n'ont que des composantes -1,0 and 1 et soit $q \in Q$ un état de contrôle. Est-ce qu'il n'existe pas une séquence de transition $(q_0, \vec{v}_0) \rightsquigarrow (q, \vec{v})$ pour un $\vec{v}$?

Nous pouvons montrer que ce problème est EXPSPACE-difficile dans la dimension du SAVE. Ce fait se déduit d'un résultat de Lipton [Lip76], qui montre essentiellement qu'il existe un réseau de Petri de dimension n tel que les vecteurs dans A ont seulement des composantes de valeurs -1,0 et 1, qui peut simuler correctement un compteur de valeur 0 à 2^{2^n}. Par conséquent, pour chaque machine de Turing qui utilise un espace borné par 2^n nous pouvons construire un SAVE tel que q n'est pas atteignable si et seulement si la machine de Turing accepte. Cela peut être fait en simulant la machine de Turing par une machine à 3 compteurs comme dans [RY86]. Avec les résultats dans la section 3.1.3 nous obtenons:

Théorème 3.6 :

> Problème 3.3 est EXPSPACE-difficile dans la dimension du SAVE.

Maintenant, nous pouvons prouver le théorème suivant où la taille d'un BPP est définie comme le nombre de dimensions.

Théorème 3.7 :

> Le problème du model-checking BPP par rapport à LTL est EXPSPACE-difficile dans la taille du système.

tel-00004890, version 1 - 19 Feb 2004

Preuve:
Nous montrons que le Problème 3.3 est réductible en temps polynômial au problème de model-checking des BPP par rapport à LTL. Nous construisons à partir d'un SAVE donné un BPP $\mathcal{P}$ et une formule φ de LTL, tel que $\mathcal{P}$ satisfait φ ssi l'état de contrôle q dans le SAVE ne peut pas être atteint.

Posons un SAVE $\mathcal{S} = (\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ à k dimensions tel que les vecteurs de l'ensemble d'addition A n'ont que de composantes de valeur $\Leftrightarrow 1$,0 et 1 et soit q un état de contrôle. Les étiquettes des transitions ne sont pas importantes pour notre construction. Nous pouvons supposer que toutes les transitions sont étiquetés différemment. Les vecteurs des transitions dans $\mathcal{S}$ peuvent contenir plus qu'une composante avec valeur $\Leftrightarrow 1$. Dans un processus BPP, chaque transition contient au plus un $\Leftrightarrow 1$. Nous divisons donc chaque transition τ contenant m composantes de valeur $\Leftrightarrow 1$ en m transitions, telles que chacune de ces transitions contient seulement une composante $\Leftrightarrow 1$ et tel que l'effet global de l'exécution de ces transitions est le même que celui de τ.

Par exemple, $q_1 \rightarrow (q_2, (\Leftrightarrow 1, \Leftrightarrow 1, \Leftrightarrow 1, 1, 0))$ est remplacée par $q_1 \rightarrow (q_3, (\Leftrightarrow 1, 0, 0, 1, 0))$, $q_3 \rightarrow (q_4, (0, \Leftrightarrow 1, 0, 0, 0)$ et $q_4 \rightarrow (q_2, (0, 0, \Leftrightarrow 1, 0, 0))$.

Les états de contrôle intermédiaires (ici q_3 et q_4) n'apparaissent pas ailleurs dans le SAVE construit. En sus, nous ajoutons une transition $q \xrightarrow{t} (q, \vec{0})$ au SAVE qui ne peut être exécutée que si q est atteignable. Appelons le SAVE ainsi construit $\mathcal{S}'$. La construction est polynômiale dans la taille de $\mathcal{S}$.

$\mathcal{S}'$ peut être obtenu comme un produit d'un processus BPP $\mathcal{P}$ et d'un automate fini $A_{\mathcal{S}'}$. L'automate $A_{\mathcal{S}'}$ représente la structure de contrôle du SAVE, donnée par les transitions sans vecteurs, tandis que le BPP $\mathcal{P}$ est donné par le SAVE, où tous les états de contrôle sont confondus en un seul. Puisque toutes les transitions sont étiquetées différemment, nous pouvons construire une formule de LTL $\varphi =$ ("chemin de $A_{\mathcal{S}'}$" $\Rightarrow \Box\neg t$) qui a comme modèle toutes les séquences d'étiquettes de transitions qui, si elles sont des chemins de l'automate $A_{\mathcal{S}'}$ ne contiennent jamais t.

Finalement, nous avons $\mathcal{P}$ satisfait φ si et seulement si il n'y a pas de séquence de transitions dans $\mathcal{S}$ tel que $(q_0, \vec{v}_0) \rightsquigarrow (q, \vec{v})$ pour un $\vec{v}$. □

Un corollaire simple du Théorème 3.7 est que le problème du model-checking de réseaux de Petri par rapport au μ-calcul linéaire est aussi EXPSPACE-difficile.

Avec les résultats des chapitres 3.3 et 3.4 nous obtenons le théorème suivant:

Théorème 3.8 (Résultat principal) :

> Le problème du *model-checking* de BPP et des réseaux de Petri par rapport à LTL ou le μ-calcul linéaire est EXPSPACE-complet dans la taille du système et PSPACE-complet dans la taille de la formule.

Remarque: Il reste un écart entre la borne inférieure qui est dans $O(2^{cn})$ et la borne supérieure qui est dans $O(2^{cnlog(n)})$. Cet écart est le même que pour le problème du caractère non-borné.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Chapitre 4

La logique temporelle linéaire contrainte

Dans ce chapitre nous introduisons la logique temporelle linéaire CLTL (*Constraint Linear Temporal Logic*). CLTL est une extension de la logique temporelle linéaire LTL introduite par Pnueli [Pnu77].

Il est bien connu que LTL définit exactement la classe des ω-langages sans hauteur d'étoile [Tho79, Wol83] et est par conséquent moins expressive que des logiques comme ETL [Wol83] et le μ-calcul linéaire. Pour obtenir le pouvoir expressif de ces logiques nous étendons LTL avec la possibilité d'ajouter des contraintes de motif, c'est-à-dire on peut exiger que la séquence à partir d'un certain point doit être incluse dans un langage fini. Par exemple, la formule $u.\Box A^u$ exprime la propriété que chaque séquence fini à partir de l'état courant (représenté par u) doit être dans le langage de A.

Nous avons choisi cette façon d'étendre LTL parce que cela nous permet d'une manière facile d'introduire les constructions pour pouvoir exprimer des propriétés non-régulières.

Pour exprimer des propriétés non-régulières nous utilisons des variables de comptages, à qui sont associées des formules propositionnelles. Ces variables sont utilisées pour définir des contraintes linéaires sur le nombre de fois les formules propositionnelles correspondantes sont satisfaites dans la séquence. Par exemple, dans la formule $[x : \pi_1, y : \pi_2]. \Diamond(x < y)$, la construction $[x : \pi_1, y : \pi_2]$ associe π_1 à x et π_2 à y. La variable x (resp. y) va compter le nombre de fois que π_1 (resp. π_2) est satisfaite dans la séquence. Et la formule est vraie s'il existe un point dans la séquence où le nombre de fois que π_2 est vraie est supérieur au nombre de fois que π_1 est vraie.

Nous appelons ALTL l'extension de LTL avec les contraintes de motif seulement. Nous montrons que ALTL a le même pouvoir expressif que le μ-calcul propositionnel linéaire (ou les ω-automates de Büchi). Nous appelons l'extension de LTL avec seulement les contraintes de comptages PLTL (Presburger LTL).

Nous montrons que le problème de satisfaisabilité de CLTL (et PLTL) est indécidable (Σ_1^1-dur) et le problème de vérification de PLTL est indécidable même pour les systèmes finis. Pour obtenir un fragment décidable nous définissons un fragment syntaxique de CLTL, appelé $\text{CLTL}_\Box$, et le fragment correspondant de PLTL, appelé $\text{PLTL}_\Box$. Ces fragments

ne sont pas fermés par négation. Nous caractérisons donc d'une manière syntaxique les fragments complémentaires de $\mathrm{CLTL}_\Box$ et $\mathrm{PLTL}_\Box$ en introduisant $\mathrm{CLTL}_\Diamond$ et $\mathrm{PLTL}_\Diamond$. Les fragments $\mathrm{CLTL}_\Box$ et $\mathrm{CLTL}_\Diamond$ sont tous les deux plus expressifs que le μ-calcul linéaire et peuvent par conséquent exprimer toutes les propriétés ω-régulières. Les fragments $\mathrm{PLTL}_\Box$ et $\mathrm{PLTL}_\Diamond$ sont plus expressifs que LTL et peuvent donc exprimer toutes les ω-propriétés sans hauteur d'étoile.

Pour ces fragments nous obtenons des résultats de décidabilité concernant les problèmes de satisfaisabilité, de validité et de vérification. Le théorème principal qui nous permet de déduire ces résultats est que les propriétés de $\mathrm{CLTL}_\Diamond$ peuvent être décomposées en des propriétés ω-régulières et des *propriétés d'inévitabilité de comptage*, c.-à-d. des contraintes sur l'ensemble de préfixes. Le même théorème est vrai pour $\mathrm{PLTL}_\Diamond$ et les ω-propriétés sans hauteur d'étoile. Avec cette décomposition nous pouvons transformer le problème de la vérification vers un *problème du vide contraint*, c.-à-d. le problème d'existence d'un préfixe dans un certain ensemble qui satisfait une contrainte. Nous étudions la décidabilité de ce problème et les techniques qui peuvent être appliquées pour le résoudre selon des propriétés et des systèmes considérés.

Nous considérons les systèmes définis dans le chapitre 2. La première méthode que nous considérons est la réduction du problème vers le problème du vide d'un ensemble semilinéaire. Cette réduction est possible si l'intersection du ω-langage du système avec la partie ω-régulière de la propriété est semilinéaire. Cela est le cas pour les automates à pile par exemple. Leur problème de la vérification par rapport à $\mathrm{CLTL}_\Box$ est donc décidable. Le problème de la vérification est par conséquent décidable pour les processus BPA. De plus, nous montrons que cette réduction n'est pas possible pour toute la classe des systèmes semilinéaires. En effet, nous montrons qu'elle ne peut pas être appliquée pour les BPP. Nous montrons que pour les processus PA, qui sont semilinéaires (voir le Théorème 2.9), le problème de la vérification par rapport à LTL est en fait indécidable. Par conséquent nous introduisons une autre paire de fragments: $\mathrm{PLTL}_\Box$ simple et $\mathrm{PLTL}_\Diamond$ simple et nous montrons que le problème de la vérification de $\mathrm{PLTL}_\Box$ simple par rapport à des processus de PA est décidable.

Puisque les réseaux de Petri ne sont pas semilinéaires nous ne pouvons pas appliquer la technique de réduction vers le problème du vide d'un ensemble semilinéaire. Mais nous pouvons réduire le problème de la vérification de $\mathrm{CLTL}_\Box$ vers le problème d'atteignabilité dans les réseaux de Petri. Le problème de la vérification de $\mathrm{CLTL}_\Box$ par rapport aux processus BPP est donc aussi décidable.

Les résultats présentés dans ce chapitre ont été publiés dans [BEH95] et [BH96].

4.1 La logique CLTL

Dans cette section nous définissons la logique CLTL (*Constraint Linear Temporal Logic*).

CLTL est une extension de la logique temporelle linéaire LTL [Pnu77] avec la possibilité d'exprimer des *contraintes de motif* et des *contraintes de comptage* sur des calculs

tel-00004890, version 1 - 19 Feb 2004

(séquences de transitions). Les contraintes de motif sont exprimées en utilisant les automates finis. Ils permettent de dire que le calcul à partir d'un point dans le passé donné correspond à un certain motif. Ce motif contraint par exemple l'ordre d'apparence d'une certaine transition (action, événement). Les contraintes de comptage sont exprimées en utilisant des formules de l'arithmétique de Presburger. Elles permettent de dire que le *nombre d'occurrences* de certaines transitions dans le calcul après un point fixé satisfait des contraintes arithmétiques spécifiées par la formule.

4.1.1 Syntaxe

Nous commençons avec la définition de la logique LTL (*Linear Temporal Logic*) introduit par Pnueli [Pnu77]. Soit $\mathcal{P}$ un ensemble fini de propositions de base.

Définition 4.1 (Formules de LTL) :

L'ensemble de formules de LTL est donné par:

$$\psi \ ::= \ P \in \mathcal{P} \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \psi\mathcal{U}\psi$$

Un ingrédient de base de CLTL sont les formules propositionnelles composées de proposition de bases et des opérateurs booléens.

Définition 4.2 (Formules propositionnelles) :

L'ensemble de formules propositionnelles est donné par:

$$\pi ::= P \in \mathcal{P} \mid \neg\pi \mid \pi \vee \pi$$

Définition 4.3 (Formules de CLTL) :

L'ensemble de formules de CLTL est donné par:

$$\varphi ::= P \in \mathcal{P} \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi\mathcal{U}\varphi \mid \widetilde{\exists} x.\varphi \mid [x : \pi].\varphi \mid f \mid u.\varphi \mid A^u$$

Nous définissons aussi deux sous-classes de CLTL, en considérant l'extension de LTL avec seulement les contraintes de motif soit les contraintes de comptage. La première sous-classe s'appelle ALTL (Automates + LTL)

Définition 4.4 (ALTL) :

L'ensemble de formules de ALTL est donné par:

$$\psi \ ::= \ P \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \psi\mathcal{U}\psi \mid u.\psi \mid A^u$$

tel-00004890, version 1 - 19 Feb 2004

La deuxième sous-classe s'appelle PLTL (Presburger LTL).

Définition 4.5 (PLTL) :

L'ensemble de formules de PLTL est donné par

$$\psi \ ::= \ P \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \psi\mathcal{U}\psi \mid \widetilde{\exists}x.\varphi \mid [x:\pi].\psi \mid f$$

Notation 4.1 :

Nous utilisons les abréviations suivantes:

$$\begin{array}{rcl}
true & = & \neg P \vee P \\
false & = & \neg true \\
\varphi_1 \wedge \varphi_2 & = & \neg(\varphi_1 \vee \varphi_2) \\
\varphi_1 \Rightarrow \varphi_2 & = & \neg\varphi_1 \vee \varphi_2 \\
\widetilde{\forall}x.\varphi & = & \neg\widetilde{\exists}x.\neg\varphi_1 \\
\Diamond\varphi & = & true\mathcal{U}\varphi \\
\Box\varphi & = & \neg\Diamond\neg\varphi \\
\varphi_1\overline{\mathcal{U}}\varphi_2 & = & \varphi_1\mathcal{U}(\varphi_1 \wedge \varphi_2) \\
[\vec{x}:\vec{\pi}].\varphi & = & [x_1:\pi_1].\cdots[x_n:\pi_n].\varphi \\
[x_1,\ldots,x_n:\pi_1,\ldots,\pi_n].\varphi & = & [x_1:\pi_1].\cdots[x_n:\pi_n].\varphi \\
[x_i:\pi_i]_{i=1}^{n}.\varphi & = & [x_1:\pi_1].\cdots[x_n:\pi_n].\varphi
\end{array}$$

Dans la formule $[x:\pi].\ \varphi$ (resp $u.\varphi$), la construction "$[x:\pi].$" (resp. "$u.$") lie la variable x (resp. u) dans la formule φ.

Une variable $x \in \mathcal{V}$ peut être liée par soit $\widetilde{\exists}$, soit la quantification de l'arithmétique de Presburger ou par la construction "$[x:\pi].$". Une variable de position $u \in \mathcal{W}$ peut être liée par la construction "$u.$". Nous appelons "$[x:\pi].$" (resp. "$u.$") la *quantification de remise à zéro* (resp. *quantification de position*).

Nous supposons sans perte de généralité que chaque variable est liée au moins une fois. Chaque variable dans une formule est soit *liée* soit *libre*.

Définition 4.6 ($\mathcal{F}(\varphi)$) :

Soit φ une formule. Nous écrivons $\mathcal{F}(\varphi)$ pour l'ensemble de variables libres de φ.

Définition 4.7 (formule fermée, ouverte) :

Une formule φ est *fermée* si toutes ses variables sont liées, sinon φ est *ouverte*.

tel-00004890, version 1 - 19 Feb 2004

4.1.2 Sémantique

Les formules de CLTL sont interprétés sur des séquences infinies sur Σ. Les opérateurs $\bigcirc, \mathcal{U}, \Diamond$, et $\Box$, sont les opérateurs *next*, *until*, *eventually*, et *always* de LTL; $\overline{\mathcal{U}}$ est l'opérateur *until fermé à droite*.

L'opérateur $\widetilde{\exists}$ est la quantification sur les entiers positifs. Nous distinguons entre $\widetilde{\exists}$ et le quantificateur $\exists$ de l'arithmétique de Presburger puisqu'ils n'ont pas la même portée. La construction "$[x : \pi]$." introduit une variable de comptage x à laquelle est associée la formule propositionnelle π. La variable x compte à partir de la position courante le nombre d'étiquettes de transitions qui satisfont π. x peut être utilisé ensuite dans des formules de Presburger f pour exprimer des *contraintes de comptage* (qui peuvent contenir plusieurs variables de comptage). Par exemple, la formule ϕ_1:

$$\phi_1 = [x, y : \pi_1, \pi_2].\Box(P \Rightarrow (x \leq y))$$

exprime le fait qu'à partir de maintenant, à chaque fois que P est vrai, le nombre de transitions qui satisfont π_2 est plus grand que le nombre de transitions qui satisfont π_1.

La construction "u." associe à la *variable de position* u la position courante sur la séquence. La variable u est utilisé comme étiquette pour faire référence à sa position associée. u peut être utilisée pour exprimer des *contraintes de motif* A^u qui dit que la sous-séquence à partir de la position u doit être acceptée par l'automate A. Par exemple, la formule ϕ_2

$$\phi_2 = u.[x, y : \pi_1, \pi_2].\Box(A^u \Rightarrow (x \leq y))$$

exprime le fait qu'à partir de maintenant, dans chaque sous-séquence finie acceptée par A, le nombre de transitions qui satisfont π_2 est plus grand ou égal que le nombre de transitions qui satisfont π_1.

La sémantique formelle de CLTL est définie en utilisant une relation de satisfaction $\models$ entre séquences de Σ^ω (avec $\Sigma = 2^{\mathcal{P}}$), des positions (entiers positifs) et des formules.

Parce que des formules peuvent être ouvertes, la relation est paramétrée par une évaluation E de variables de $\mathcal{V}$, une fonction θ qui associe à chaque variable de position ou de comptage la position où elle était introduite, et une fonction η qui associe à chaque variable de comptage la formule propositionnelle correspondante.

Soit ξ égal à E, θ, ou η. Alors, nous écrivons $\mathcal{D}(\xi)$ pour le domaine de ξ. La fonction ξ telle que $\mathcal{D}(\xi) = \emptyset$ est appelée $\emptyset$. Nous écrivons $\xi[z \leftarrow \kappa]$ pour la fonction ξ' telle que $\mathcal{D}(\xi') = \mathcal{D}(\xi) \cup \{z\}$, qui associe la valeur κ à z et coïncide avec ξ sur toutes les autres variables.

Définition 4.8 (relation de satisfaction) :

Soit $\sigma \in \Sigma^\omega$. Pour chaque $i \geq 0$, chaque évaluation E, chaque θ (tel que $\forall z \in \mathcal{D}(\theta)$, $0 \leq \theta(z) \leq i$) et chaque η, nous définissons $\langle \sigma, i \rangle \models_{(E,\theta,\eta)} \varphi$ inductivement sur la structure de φ comme suit:

tel-00004890, version 1 - 19 Feb 2004

$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} P$	ssi	$P \in \sigma(i)$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} \neg\varphi$	ssi	$\langle \sigma, i\rangle \not\models_{(E,\theta,\eta)} \varphi$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} \varphi_1 \vee \varphi_2$	ssi	$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} \varphi_1$ ou $\langle \sigma, i\rangle \models_{(E,\theta,\eta)} \varphi_2$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} \bigcirc\varphi$	ssi	$\langle \sigma, i+1\rangle \models_{(E,\theta,\eta)} \varphi$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} \varphi_1 \mathcal{U} \varphi_2$	ssi	$\exists j.\ i \leq j.\ \langle \sigma, j\rangle \models_{(E,\theta,\eta)} \varphi_2$ et $\forall k.\ i \leq k < j.\ \langle \sigma, k\rangle \models_{(E,\theta,\eta)} \varphi_1$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} \widetilde{\exists} x.\varphi$	ssi	$\exists k \in I\!N.\ \langle \sigma, i\rangle \models_{(E',\theta,\eta)} \varphi$ où $E' = E[x \leftarrow k]$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} [x : \pi].\varphi$	ssi	$\langle \sigma, i\rangle \models_{(E,\theta',\eta')} \varphi$ où $\theta' = \theta[x \leftarrow i]$ et $\eta' = \eta[x \leftarrow \pi]$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} f$	ssi	$E' \models f$ où $E' = E[x \leftarrow \lvert\{j \in [\theta(x), i] \ : \ \langle \sigma, j\rangle \models \eta(x)\}\rvert]_{x \in \mathcal{D}(\eta)}$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} u.\varphi$	ssi	$\langle \sigma, i\rangle \models_{(E,\theta',\eta)} \varphi$ où $\theta' = \theta[u \leftarrow i]$
$\langle \sigma, i\rangle \models_{(E,\theta,\eta)} A^u$	ssi	$\sigma(\theta(u), i) \in L(A)$

Soit φ une formule fermée. Il est clair que $\langle \sigma, i\rangle \models_{(E,\theta,\eta)} \varphi$ si et seulement si $\langle \sigma, i\rangle \models_{(\emptyset,\emptyset,\emptyset)} \varphi$, et par conséquent nous écrivons simplement $\langle \sigma, i\rangle \models \varphi$. Nous écrivons aussi $\sigma \models \varphi$ et nous disons que σ *satisfait* φ, si $\langle \sigma, 0\rangle \models \varphi$.

Définition 4.9 (Sémantique d'une formule fermée) :

Soit φ une formule fermée. Alors $[\![\varphi]\!] := \{\sigma \in \Sigma^\omega \ : \ \sigma \models \varphi\}$.

Définition 4.10 (Équivalence de deux formules) :

Soient φ_1 et φ_2 deux formules fermées. Elles sont *équivalentes*, représenté par $\varphi_1 = \varphi_2$, ssi $[\![\varphi_1]\!] = [\![\varphi_2]\!]$.

Définition 4.11 (satisfaisable, valide) :

Pour chaque $S \subseteq \Sigma^\omega$, φ est *satisfaisable* (resp. *valide*) relative à S ssi $S \cap [\![\varphi]\!] \neq \emptyset$ (resp. $S \subseteq [\![\varphi]\!]$), et φ est *satisfaisable* (resp. *valide*) ssi elle est satisfaisable (resp. valide) relative à Σ^ω.

Définition 4.12 (Problème de vérification) :

Étant donné une formule φ est un $S \subseteq \Sigma^\omega$, le problème de vérification est de savoir, si φ est valide relative à S.

4.2 Expressivité

Dans cette section nous étudions l'expressivité de la logique CLTL. Nous considérons d'abord l'expressivité de CLTL par rapport aux propriétés régulières et ensuite par rapport aux propriétés non-régulières.

tel-00004890, version 1 - 19 Feb 2004

4.2.1 Propriétés régulières

Nous montrons que ALTL a le même pouvoir expressif que les ω-automates. Pour cela nous montrons entre autre qu'une formule de ALTL peut être traduite vers une formule équivalente (modulo projection) d'une version de ETL [Wol83] (Extended Temporal Logic). Nous donnons d'abord la définition de ETL_l.

Définition 4.13 (formule de ETL_l) :

Soit $\mathcal{P}$ un ensemble de propositions atomiques. L'ensemble de formules de ETL_l est donné par:

$$\varphi ::= P \in \mathcal{P} \mid \neg\varphi \mid \varphi \vee \varphi \mid A(\varphi_1, \dots, \varphi_n)$$

où $A(\varphi_1, \dots, \varphi_n)$ (appelé *automate ETL_l*) est un système de transitions étiquetées fini $(Q, \{\varphi_1, \dots, \varphi_n\}, q_0, \delta)$.

Définition 4.14 (relation de satisfaction de ETL_l) :

Soient $\sigma \in \Sigma^\omega$ et φ une formule de ETL_l. Alors nous définissons $\langle\sigma, i\rangle \models \varphi$ inductivement comme suit:

$\langle\sigma, i\rangle \models P$	ssi	$P \in \sigma(i)$
$\langle\sigma, i\rangle \models \neg\varphi$	ssi	$\langle\sigma, i\rangle \not\models \varphi$
$\langle\sigma, i\rangle \models \varphi_1 \vee \varphi_2$	ssi	$\langle\sigma, i\rangle \models \varphi_1$ ou $\langle\sigma, i\rangle \models \varphi_2$
$\langle\sigma, i\rangle \models A(\varphi_1, \dots, \varphi_n)$	ssi	$A(\varphi_1, \dots, \varphi_n)$ a un calcul l-accepteur sur $\sigma(i)\sigma(i+1)\cdots$

Un *calcul l-accepteur* de $A(\varphi_1, \dots, \varphi_n)$ sur σ est un calcul $\rho \in Q^\omega$ de $A(\varphi_1, \dots, \varphi_n)$ tel que $\forall i \in I\!N$ si $(q_i, \varphi_j, q_{i+1}) \in \delta$ alors $\langle\sigma, i\rangle \models \varphi_j$.

Cette variante de ETL s'appelle ETL_l parce que la définition d'un calcul accepteur est la *looping acceptance*, c.-à-d. un calcul est accepteur s'il est infini. Les notions de satisfaisabilité, sémantique d'une formule etc. sont définies comme pour CLTL. Nous avons le théorème suivant [VW94]:

Théorème 4.1 :

Pour chaque formule φ de ETL_l il existe un automate de Büchi $\mathcal{A}$ tel que $[\![\varphi]\!] = L(\mathcal{A})$ et vice versa.

Théorème 4.2 :

Soit φ une formule fermée de ALTL avec un ensemble de propositions atomiques $\mathcal{P}$. Soit $\Sigma = 2^{\mathcal{P}}$. Alors, il existe une formule φ' de ETL_l telle que $[\![\varphi]\!] = [\![\varphi']\!]|_\Sigma$

tel-00004890, version 1 - 19 Feb 2004

Preuve:
Soit φ une formule de ALTL. Soit $\{A_1, \ldots, A_n\}$ l'ensemble d'automates finis qui sont utilisés dans φ. Nous supposons que les automates A_i sont complets (voir le Lemme 2.6). En plus nous supposons que pour chaque variable de position u il y a exactement un automate A_i associé tel que A_i^u apparaît dans la formule. D'abord nous observons que les opérateurs $\mathcal{U}$ et $\bigcirc$ peuvent facilement être traduit vers des formules de ETL$_l$ (voir [VW94]). Ils restent les construction $u.\varphi$ et A^u à traduire. Nous donnons d'abord l'idée intuitive: Nous ajoutons des proposition atomiques à $\mathcal{P}$ qui indiquent quand un des automates entre dans un état final. Nous transformons les automates A_i ainsi. Ensuite pour chaque $u.\varphi$ nous "lançons" l'automate qui fait référence à u. Les propositions A^u dans φ sont remplacées par une des nouvelles propositions correspondantes. Formellement:

Soient $\mathcal{P}' = \mathcal{P} \cup \{fin(A_1), \ldots, fin(A_n)\}$ et $\Sigma' = 2^{\mathcal{P}'}$. Pour chaque automate $A_i = (Q_i, \Sigma, q_0^i, \delta_i, F_i)$ nous construisons un automate ETL_l $A_i'(\Sigma') = (Q_i, \Sigma', q_0^i, \delta_i')$ où δ_i' est le plus petit ensemble qui satisfait:

- Si $(q_1, a, q_2) \in \delta_i$ et $q_2 \notin F_i$, alors $\forall a' \in \tilde{a}.\ (q_1, a' \setminus \{fin(A_i)\}, q_2) \in \delta_i'$
- Si $(q_1, a, q_2) \in \delta_i$ et $q_2 \in F_i$, alors $\forall a' \in \tilde{a}.\ (q_1, a' \cup \{fin(A_i)\}, q_2) \in \delta_i'$

Maintenant, pour obtenir une formule de ETL$_l$ nous remplaçons chaque $u.\varphi_1$ dans φ par $A_i'(\Sigma') \wedge \varphi_1$, où A_i' est l'automate ETL$_l$ construit à partir de l'automate A_i associé a u. Ensuite, chaque formule de la forme A_i^u dans φ est remplacé par la proposition $fin(A_i)$. Nous appelons φ' la formule ETL$_l$ ainsi obtenu. Il est facile de voir que $[\![\varphi]\!] = [\![\varphi']\!]|_\Sigma$. En effet, les nouvelles propositions $fin(A_i)$ imposent qu'à chaque point où la proposition A_i^u apparaît, la séquence entre le point u et ce point doit être dans le langage de A_i. Puisque les automates A_i sont complets et grâce à la construction des A_i', il n'y a pas d'autres contraintes imposées dans φ' pour les séquences à partir du point u. □

Théorème 4.3 :

Pour chaque formule fermée de ALTL φ il existe un automate de Büchi $\mathcal{A}$ avec $[\![\varphi]\!] = L(\mathcal{A})$ et vice versa.

Preuve:
Nous montrons d'abord que pour chaque automate de Büchi il existe une formule de ALTL équivalente. D'après le théorème de McNaughton [Tho90] chaque langage ω-régulier peut être défini par une formule de ALTL de la forme

$$u.\ \bigvee_{i=1}^{n} (\Box\Diamond A_i^u \wedge \Diamond\Box B_i^u)$$

Nous montrons ensuite que chaque formule de ALTL peut être décrite par un automate de Büchi. Pour chaque formule fermée de ALTL φ, nous pouvons construire un ω-automate de

tel-00004890, version 1 - 19 Feb 2004

Büchi qui reconnaît $[\![\varphi]\!]$ en construisant d'abord la formule de ETL_l φ' avec $[\![\varphi]\!] = [\![\varphi']\!]|_\Sigma$ en utilisant le Théorème 4.2. Ensuite nous construisons l'automate de Büchi $\mathcal{A}'$ équivalent à φ' par le Théorème 4.1. Finalement nous obtenons un automate de Büchi $\mathcal{A}$ avec $L(\mathcal{A}) = [\![\varphi]\!]$ en projetant $\mathcal{A}'$ sur Σ.

□

La logique ALTL peut donc exprimer toutes les propriétés ω-régulières comme la logique ETL [Wol83] *(extended temporal logic)* ou le μ-calcul linéaire μTL [Var88].

4.2.2 Propriétés non-régulières

En utilisant les contraintes de comptages nous pouvons exprimer dans CLTL des propriétés *non-régulières*, i.e. des propriétés qui ne peuvent pas être exprimées par les automates ω-réguliers.

Par exemple, considérons la propriété suivante:

Propriété 4.1 :

> *Étant donné une séquence infinie de transitions (événements), chaque a est suivi par un b, et à chaque position entre un a et le prochain b le nombre de c et plus grand ou égal au nombre de d et en plus, à b le nombre de c et de d est égal.*

Formellement, la propriété impose que les séquences entre deux a et b successifs soient dans le langage

$$\{\sigma \in \Sigma^* \ : \ |\sigma|_c = |\sigma|_d \text{ et } \forall i \leq |\sigma|, \ |\sigma(0,i)|_c \geq |\sigma(0,i)|_d\}$$

Cette propriété peut être exprimée par la formule PLTL:

$$\Box\,(a \Rightarrow [x,y,z : c,d,b].\,((x \geq y\,\mathcal{U}\,b) \wedge \Box\,((b \wedge z = 1) \Rightarrow x = y))) \tag{4.1}$$

L'introduction des contraintes de comptage permet de caractériser des langages non-réguliers qui peuvent être hors-contextes comme dans (4.1) mais aussi dépendant du contexte si nous considérons des contraintes qui lient plus que 2 variables de comptages.

L'introduction des contraintes de motifs permet de contraindre l'ordre d'apparence d'événements. Supposons par exemple que nous voulons renforcer la propriété 4.1 en imposant qu'entre deux a et b successifs, tous les c apparaissent avant les d.

Propriété 4.2 :

> *Étant donnée une séquence infinie de transitions (événements), chaque a est suivi par un b, et à chaque position entre un a et le prochain b, le nombre de c est plus grand ou égal au nombre de d et, à b le nombre de c et de d est égal. Entre deux a et b successifs, tous les c apparaissent avant les d.*

tel-00004890, version 1 - 19 Feb 2004

Cette nouvelle propriété peut être exprimée par la conjonction de la formule de LTL $\Box(a \Rightarrow \Diamond b)$ (chaque a est suivi d'un b) avec la formule de CLTL:

$$u.\,[x, y : c, d].\ \Box\,(A^u \Rightarrow (B^u \wedge x = y)) \tag{4.2}$$

où A et B sont des automates d'états finis, tels que

$$L(A) = \Sigma^* a(\Sigma \Leftrightarrow \{a, b\})^* b$$

et

$$L(B) = \Sigma^* a(\Sigma \Leftrightarrow \{d\})^* (\Sigma \Leftrightarrow \{c\})^* b$$

La propriété 4.2 peut être exprimée sans contrainte de motif puisque la contrainte de motif utilisée peut être exprimée en LTL. En fait, cette propriété correspond à la conjonction de la formule de LTL

$$\Box\,(a \Rightarrow \bigcirc\,(\neg d\,\mathcal{U}\,\neg c\,\mathcal{U}\,b))$$

avec la formule (4.1). Puisque LTL peut seulement exprimer les ω-propriétés sans hauteur d'étoile [Tho79, Wol83] et puisque les contraintes de comptages ne peuvent pas contraindre l'ordre entre événements, l'utilisation de contraintes de motifs enrichit l'expressivité de PLTL.

Finalement, nous donnons un exemple de l'utilisation de la quantification $\widetilde{\forall}$. Cette quantification permet d'une part de mettre en relation des contraintes de comptage à des positions différentes de la séquence et permet d'autre part d'exprimer des contraintes de comptages sur le nombre d'occurrences d'événements dans des sous-séquences différentes. Par exemple, considérons la propriété suivante:

Propriété 4.3 :

Chaque a est suivi d'un b, et entre deux a et b successif les sous-séquences sont dans le langage

$$\{\sigma s \sigma' \ :\ \sigma, \sigma' \in (\Sigma \Leftrightarrow \{a, b, s\})^*, |\sigma|_c = |\sigma'|_d\}$$

La Propriété 4.3 peut être exprimée par la conjonction de la formule de LTL:

$$\Box\,(a \Rightarrow \bigcirc\,(\neg s\,\mathcal{U}\,(s \wedge \bigcirc\,(\neg s\,\mathcal{U}\,b))))$$

avec la formule de PLTL:

$$\widetilde{\forall} n.\ \Box\,(a \Rightarrow [x, y, z : c, d, b].\ \Box\,((s \wedge x = n) \Rightarrow \Box\,((b \wedge z = 1) \Rightarrow y = n))) \tag{4.3}$$

La variable n dans (4.3) est utilisée pour mémoriser le nombre de c entre a et s et ce nombre est comparé ensuite avec le nombre de d entre s et b.

tel-00004890, version 1 - 19 Feb 2004

4.3 Résultats d'indécidabilité pour CLTL

Dans cette section nous montrons des résultats d'indécidabilité concernant la logique CLTL et ses sous-classes. Tous ces résultats sont basés sur le fait que dans CLTL on peut exprimer des propriétés d'invariance de contraintes qui permettent de simuler les tests sur zéro d'une machine à deux compteurs (machine de Minsky). Nous donnons d'abord la définition d'une machine à deux compteurs. Il est bien connu qu'une machine à deux compteurs peut simuler une machine de Turing.

Définition 4.15 (Machine à deux compteurs) :

Une machine à deux compteurs $\mathcal{M}$ est un programme d'états finis avec 2 compteurs c_1 et c_2 et $m \in I\!N$ instructions étiquetées

$$\begin{array}{lcl} s_1 & : & com_1, \\ s_2 & : & com_2, \\ \vdots & & \vdots \\ s_{m-1} & : & com_{m-1}, \\ s_m & : & halt. \end{array}$$

Les instructions com_i ont la forme

- $c_i := c_i + 1;\ goto\ s_k$ où
- $if\ c_i = 0\ then\ goto\ s_{k_1}\ sinon\ c_i := c_i \Leftrightarrow 1;\ goto\ s_{k_2}$

L'exécution d'une machine à deux compteurs $\mathcal{M}$ commence avec com_1, où les deux compteurs ont la valeur 0 et parcourt les instructions comme indiqué par les instructions *goto*. L'exécution s'arrête si l'état *halt* est atteint.

Nous pouvons d'abord montrer que le problème de satisfaisabilité de PLTL et CLTL est “hautement” indécidable. Une définition de l'hiérarchie arithmétique peut être trouvé dans [Rog87].

Théorème 4.4 :

Les problèmes de satisfaisabilité de PLTL et CLTL sont Σ_1^1-complets.

Preuve:

- La satisfaisabilité d'une formule φ de CLTL peut être exprimée en Σ_1^1. La formule Σ_1^1 exprime l'existence d'une séquence qui satisfait la formule φ. Chaque séquence infinie peut être codée par un nombre fini d'ensembles infinis d'entiers qui indiquent à quel point dans la séquence les propositions atomiques sont satisfaites. La relation de satisfaction peut être exprimée par un prédicat du premier ordre.

tel-00004890, version 1 - 19 Feb 2004

– Pour montrer que le problème est Σ_1^1-difficile nous montrons que nous pouvons coder le problème de calcul récurrent (c'est-à-dire le problème d'existence d'un calcul qui passe infiniment souvent par l'état initial de la machine) d'une machine à 2 compteurs non-déterministe comme la satisfaisabilité d'une formule PLTL. [HPS83] et [AH89] ont prouvé que le problème de calcul récurrent est Σ_1^1-difficile. Étant donnée une machine à deux compteur $\mathcal{M}$, nous construisons une formule $([x_i, y_i : inc_i, dec_i]_{i=1}^2 \Box\varphi) \wedge \Box\Diamond P$ où φ code la relation de transition de $\mathcal{M}$ (le test sur zéro est fait en comparant les deux variables de comptages x_i et y_i; inc_i correspond à l'action $c_i := c_i + 1$ et dec_i à l'action $c_i := c_i \Leftrightarrow 1$ et P est une proposition atomique qui correspond à l'état initial de la machine.

□

Une conséquence immédiate du théorème 4.4 est le corollaire suivant.

Corollaire 4.1 :

Les problèmes de validité de PLTL et CLTL sont Π_1^1-complet.

Théorème 4.5 :

Le problème de vérification de système de transitions étiquetés fini par rapport à PLTL et CLTL est Π_1^1-complets.

Preuve:
Ce théorème se déduit immédiatement du théorème 4.4 puisque Σ^ω peut être généré par un système de transitions étiquetées fini. □

Nous pouvons prouver l'indécidabilité du problème de vérification pour une classe très simple de formules de PLTL, les *propriétés d'inévitabilité de comptage*.

Définition 4.16 (Propriété d'inévitabilité de comptage) :

Une *propriété d'inévitabilité de comptage* est une formule de la forme:

$$[\vec{x} : \vec{\pi}].\ \Diamond f(\vec{x}) \tag{4.4}$$

Proposition 4.1 :

Le problème de vérification de système de transitions étiquetées fini par rapport à des propriétés d'inévitabilité de comptage est Σ_1^0-complet.

tel-00004890, version 1 - 19 Feb 2004

Preuve:

– Puisque les systèmes de transitions étiquetées finis sont à branchement fini et les contraintes de comptage dépendent seulement du passé, la vérification d'une formule $\Diamond f$ pour de tels systèmes est semi-décidable (dans Σ_1^0).

– Nous montrons que le problème de vérification est Σ_1^0-difficile en réduisant le problème de l'arrêt d'une machine à deux compteurs déterministe. La structure de contrôle d'une machine à deux compteurs peut être vue comme un système de transitions étiquetées fini $\mathcal{S}$ avec les étiquettes inc_i, dec_i, z_i $(i = 1, 2)$ où inc_i correspond à $c_i := c_i + 1$, dec_i à $c_i := c_i \Leftrightarrow 1$ et z_i à $c_i = 0$. Nous ajoutons en plus une boucle à l'état d'arrêt. La sémantique d'un système est donnée par toutes ses séquences infinies. La formule f décrit toutes les séquences de $\mathcal{S}$ qui sont un *mauvais calcul* de la machine à deux compteurs ou qui atteignent l'état d'arrêt. Soit φ donnée par:

$$([x_i, y_i : inc_i, dec_i]_{i=1}^2. \Diamond((z_i \wedge (x_i \Leftrightarrow y_i) \neq 0) \vee a_i)$$

où a_i est la transition qui mène vers l'état d'arrêt. Les propositions atomiques z_i et a_i peuvent être codées par des compteurs pour obtenir une formule de la forme (4.4). Par exemple pour chaque transition $q_1 \stackrel{a_i}{\to} q_2$ nous ajoutons deux états nouveaux q_1' et q_2' avec $q_1 \stackrel{ia_i}{\to} q_1' \stackrel{a_i}{\to} q_2' \stackrel{da_i}{\to} q_2$. $|ia_i| \Leftrightarrow |da_i| = 1$ signifie que la prochaine transition doit être un a_i. La proposition a_i peut donc être codée par des compteurs.

Soit S l'ensemble de toutes les séquences de calcul infini de $\mathcal{S}$. Alors nous avons φ est valide relative à S si et seulement si la machine à deux compteurs s'arrête.

□

Pour obtenir des résultats de décidabilité pour le problème de la vérification nous devons par conséquent interdire des formules de la forme (4.4). Pour cela nous introduisons des fragments de CLTL dans la section suivante.

4.4 Les fragments de CLTL

Dans cette section nous introduisons plusieurs fragments de CLTL. Ces fragments sont définis de façon qu'ils ne contiennent pas des formules qui causent l'indecidabilité du problème de vérification, i.e. des formules de la forme $\Diamond f$. Ces fragments ne sont pas fermés par négation. Pour chacun des fragments nous introduisons un autre fragment tel que chaque formule dans le premier est équivalent à une formule du deuxième et vice versa.

4.4.1 Définitions

Nous commençons avec la définition des fragments les plus expressifs, $\text{CLTL}_\Box$ et $\text{CLTL}_\Diamond$. Pour décrire facilement les restrictions syntaxiques qui correspondent à ces fragments, nous

tel-00004890, version 1 - 19 Feb 2004

introduisons d'abord la *forme positive* de formules de CLTL donnée par:

Définition 4.17 (Forme positive de CLTL) :

Une formule en *forme positive* de CLTL est donnée par:

$$\begin{aligned} \varphi \ ::= \ & \widetilde{\exists} x.\varphi \mid \widetilde{\forall} x.\varphi \mid [x:\pi].\varphi \mid f \mid u.\varphi \mid A^u \mid P \mid \neg P \mid \\ & \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \varphi\mathcal{U}\varphi \end{aligned} \tag{4.5}$$

Lemme 4.1 (Forme positive) :

Chaque formule CLTL a une formule équivalente en forme positive.

Preuve:
Nous prouvons ce lemme par induction structurelle en utilisant les propriétés des opérateurs booléens et les équivalences de formules ($\bar{A}$ est l'automate complémentaire de A):

$$\neg[x:\pi].\,\varphi = [x:\pi].\,\neg\varphi \tag{4.6}$$
$$\neg u.\,\varphi = u.\,\neg\varphi \tag{4.7}$$
$$\neg A^u = \bar{A}^u \tag{4.8}$$
$$\neg\bigcirc\varphi = \bigcirc\neg\varphi \tag{4.9}$$
$$\neg(\varphi_1\mathcal{U}\varphi_2) = (\Box\neg\varphi_2) \vee (\neg\varphi_2\overline{\mathcal{U}}\neg\varphi_1) \tag{4.10}$$

□

Les formes positives de formules ALTL (resp. PLTL) correspondent à (4.5) sans quantification de remise à zéro (resp. position) et contraintes de comptage (resp. motif).

La Proposition 4.1 montre que le problème de vérification devient indécidable dès que des formules de la forme $\Diamond f$ sont considérées. Pour éviter ce genre de formules nous définissons le fragment CLTL$_\Box$ en imposant que dans (4.17) la partie droite de $\mathcal{U}$ doit être une formule de ALTL (c'est-à-dire sans contraintes de comptages). Nous admettons dans ce fragment aussi des formules $\overline{\mathcal{U}}$ avec la même restriction. Nous interdisons aussi dans CLTL$_\Box$ l'opérateur $\widetilde{\exists}$. Le fragment CLTL$_\Diamond$ est défini de sorte qu'il caractérise exactement les négations de formule de CLTL$_\Box$. Nous définissons ensuite les fragments correspondants pour PLTL. Dans PLTL$_\Box$ la partie à droite de $\mathcal{U}$ doit être une formule de LTL. Finalement, nous définissons les fragments PLTL$_\Box$ simple et PLTL$_\Diamond$ simple. Dans PLTL$_\Box$ simple la partie à droite de $\mathcal{U}$ doit être une formule propositionnelle.

Définition 4.18 (CLTL$_\Box$) :

Les formules de CLTL$_\Box$ sont données par:

$$\varphi \ ::= \ \widetilde{\forall} x.\varphi \mid [x:\pi].\varphi \mid f \mid u.\varphi \mid A^u \mid P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \varphi\mathcal{U}\psi \mid \varphi\overline{\mathcal{U}}\psi$$

où ψ est une formule de ALTL.

tel-00004890, version 1 - 19 Feb 2004

Définition 4.19 (CLTL$_\Diamond$) :

Les formules de CLTL$_\Diamond$ sont données par:

$$\varphi \ ::= \ \widetilde{\exists} x.\varphi \mid [x:\pi].\varphi \mid f \mid u.\varphi \mid A^u \mid P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Box\psi \mid \psi\mathcal{U}\varphi \mid \psi\overline{\mathcal{U}}\varphi$$

où ψ est une formule de ALTL.

Définition 4.20 (PLTL$_\Box$) :

Les formules de PLTL$_\Box$ sont données par:

$$\varphi \ ::= \ \widetilde{\forall} x.\varphi \mid [x:\pi].\varphi \mid f \mid P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \varphi\mathcal{U}\psi \mid \varphi\overline{\mathcal{U}}\psi$$

où ψ est une formule de LTL.

Définition 4.21 (PLTL$_\Diamond$) :

Les formules de PLTL$_\Diamond$ sont données par:

$$\varphi \ ::= \ \widetilde{\exists} x.\varphi \mid [x:\pi].\varphi \mid f \mid P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Box\psi \mid \psi\mathcal{U}\varphi \mid \psi\overline{\mathcal{U}}\varphi$$

où ψ est une formule de LTL.

Définition 4.22 (PLTL$_\Box$ simple) :

Les formules de PLTL$_\Box$ simple sont données par:

$$\varphi \ ::= \ \widetilde{\forall} x.\varphi \mid [x:\pi].\varphi \mid f \mid P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \varphi\mathcal{U}\psi \mid \varphi\overline{\mathcal{U}}\psi$$

où ψ est une formule propositionnelle.

Définition 4.23 (PLTL$_\Diamond$ simple) :

Les formules de PLTL$_\Diamond$ simple sont données par:

$$\varphi \ ::= \ \widetilde{\exists} x.\varphi \mid [x:\pi].\varphi \mid f \mid P \mid \neg P \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Box\psi \mid \psi\mathcal{U}\varphi \mid \psi\overline{\mathcal{U}}\varphi$$

où ψ est une formule propositionnelle.

Proposition 4.2 :

Pour chaque formule fermée φ de CLTL$_\Box$ (resp. PLTL$_\Box$, PLTL$_\Box$ simple) il existe une formule fermée φ' de CLTL$_\Diamond$ (resp. PLTL$_\Diamond$, PLTL$_\Diamond$ simple) telle que $[\![\neg\varphi]\!] = [\![\varphi']\!]$, et vice versa.

Preuve:
Par induction structurelle en utilisant les équivalences:

$$\neg(\varphi_1\mathcal{U}\varphi_2) = (\Box\neg\varphi_2) \vee (\neg\varphi_2\overline{\mathcal{U}}\neg\varphi_1) \quad (4.11)$$
$$\neg(\varphi_1\overline{\mathcal{U}}\varphi_2) = (\Box\neg\varphi_2) \vee (\neg\varphi_2\mathcal{U}\neg\varphi_1) \quad (4.12)$$
$$\neg\bigcirc\varphi = \bigcirc\neg\varphi \quad (4.13)$$
$$\neg\widetilde{\forall} x.\varphi = \widetilde{\exists} x.\neg\varphi \quad (4.14)$$

□

tel-00004890, version 1 - 19 Feb 2004

4.4.2 Expressivité

Il est clair que l'ensemble de formules de ALTL en forme positive est un sous-ensemble de $\mathrm{CLTL}_\Box$ et $\mathrm{CLTL}_\Diamond$. Par conséquent, $\mathrm{CLTL}_\Box$ et $\mathrm{CLTL}_\Diamond$ peuvent exprimer toutes les propriétés ω-régulières. Mais, ces deux fragments n'expriment évidemment pas les même classes de propriétés non-régulières. Par exemple, (4.2) peut être exprimée en $\mathrm{CLTL}_\Box$ mais pas sa négation.

D'une manière similaire, $\mathrm{PLTL}_\Box$ et $\mathrm{PLTL}_\Diamond$ peuvent exprimer toutes les formules de la logique LTL et peuvent donc exprimer toutes les ω-propriétés sans hauteur d'étoile. En plus, il est facile de voir que $\mathrm{PLTL}_\Diamond$ simple peut exprimer toutes les langages ω-réguliers simples et $\mathrm{PLTL}_\Box$ simple toutes leur compléments.

La même dualité qui existe entre $\mathrm{CLTL}_\Box$ et $\mathrm{CLTL}_\Diamond$ par rapport à des propriétés non-régulières existe aussi entre leurs sous-fragments. Même restreints, ses sous-fragments permettent de capturer des classes significatives de propriétés non-régulières. Par exemple, les formules (4.1) and (4.3) sont dans $\mathrm{PLTL}_\Box$ simple. Ces fragments de PLTL peuvent par ailleurs exprimer des ω-propriétés qui ne sont pas sans hauteur d'étoile en utilisant des formules de Presburger. Par exemple, la propriété qu'à chaque position paire, la propriété P est vrai [Wol83] peut être exprimer par la formule $\mathrm{PLTL}_\Box$ simple

$$[x : true].\,\Box((\exists y.\, 2y = x) \Rightarrow P) \tag{4.15}$$

La figure suivante montre les inclusions entre les logiques différentes que nous utilisons

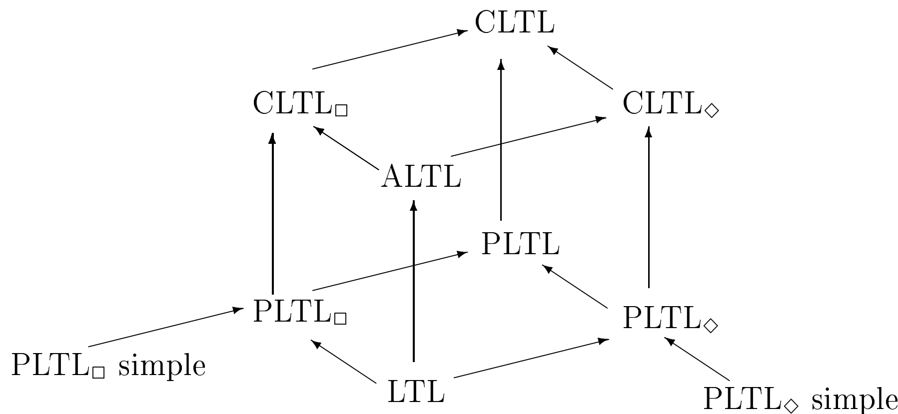


FIG. 4.1 – *Les fragments de CLTL*

4.4.3 Les problèmes de satisfaisabilité et validité pour les fragments

Nous avons les résultats suivants concernant les problèmes de satisfaisabilité et validité des fragments de CLTL introduit dans cette section.

Théorème 4.6 :

Les problèmes de satisfaisabilité de PLTL$_\Box$ simple, PLTL$_\Box$, et CLTL$_\Box$ sont Σ_1^1-complets.

Preuve:
La formule qui permet dans la preuve du théorème 4.4 de coder le problème de calcul récurrent d'une machine à 2 compteurs est une formule de PLTL$_\Box$ simple. □

Corollaire 4.2 :

Les problèmes de validité de PLTL$_\diamond$ simple, PLTL$_\diamond$, et CLTL$_\diamond$ sont Π_1^1-complets.

4.5 La décomposition de formules CLTL$_\diamond$

Nous montrons dans cette section que chaque propriété CLTL$_\diamond$ peut être décomposée (modulo projection) en une propriété ω-régulière et une propriété d'inévitabilité de comptage. Plus précisément, chaque propriété CLTL$_\diamond$ est une union finie de projections d'ensembles qui sont des intersections de propriétés ω-régulières avec des propriétés d'inévitabilité de comptage.

Cette décomposition nous permet de raisonner sur le problème de satisfaisabilité (relative) de CLTL$_\diamond$ et par conséquent sur le problème de vérification de CLTL$_\Box$, puisque CLTL$_\Box$ et CLTL$_\diamond$ sont duaux (voir Proposition 4.2).

4.5.1 Forme normale

Nous définissons d'abord une forme normale pour CLTL$_\diamond$ et nous montrons que chaque formule de CLTL$_\diamond$ est équivalente à une formule en forme normale.

Définition 4.24 (Forme normale pour CLTL$_\diamond$) :

Soit $\Sigma = \{a_1, \cdots, a_n\}$, et pour chaque $i \in \{1, \cdots, n\}$, soit $\pi_i = (\bigwedge_{P \in a_i} P) \wedge (\bigwedge_{P \notin a_i} \neg P)$. Une formule de CLTL$_\diamond$ est en *forme normale* si elle est donnée par la grammaire suivante:

$$\begin{array}{lcl} \phi & ::= & \bigvee \widetilde{\exists} \vec{x}.\ \varphi \\ \varphi & ::= & u.\ [x_i : \pi_i]_{i=1}^{n}.\ (\psi \wedge (\psi' \overline{\mathcal{U}} (f \wedge \bigcirc \varphi))) \mid u.\psi \end{array}$$

où ψ et ψ' sont des formules de ALTL sans construction "$u.$".

tel-00004890, version 1 - 19 Feb 2004

Proposition 4.3 (Forme normale de CLTL$_\diamond$) :

Pour chaque formule de CLTL$_\diamond$ il y a une formule équivalente de CLTL$_\diamond$ qui est en forme normale.

Preuve:

Nous montrons d'abord que il existe pour chaque formule une formule équivalent en forme normal sans la restriction que les $[\vec{x} : \vec{\pi}]$ qui apparaissent dans la formule doivent avoir la forme $[x_i : \pi_i]_{i=1}^n$. Nous prouvons le lemme sans cette restriction par induction sur la structure des formules CLTL$_\diamond$ en utilisant les règles suivantes:

$$\widetilde{\exists} x.(\varphi_1 \vee \varphi_2) = (\widetilde{\exists} x.\varphi_1) \vee (\widetilde{\exists} x.\varphi_2) \quad (4.16)$$

$$[x : \pi].(\varphi_1 \vee \varphi_2) = ([x : \pi].\varphi_1) \vee ([x : \pi].\varphi_2) \quad (4.17)$$

$$[\vec{x} : \vec{\pi}].\widetilde{\exists} x.\varphi = \widetilde{\exists} x.[\vec{x} : \vec{\pi}].\varphi \quad (4.18)$$

$$[\vec{x} : \vec{\pi}].u.\varphi = u.[\vec{x} : \vec{\pi}].\varphi \quad (4.19)$$

$$f = [x : true].(true\overline{\mathcal{U}}((x = 1) \wedge f)) \quad (4.20)$$

$$\psi = [x : true].(\psi\overline{\mathcal{U}}(x = 1)) \quad (4.21)$$

$$\widetilde{\exists} x.\varphi_1 \wedge \widetilde{\exists} y.\varphi_2 = \widetilde{\exists} x.\widetilde{\exists} y(\varphi_1 \wedge \varphi_2) \quad \text{si x différent de y} \quad (4.22)$$

$$u.(\varphi_1 \wedge \varphi_2) = u.\varphi_1 \wedge u.\varphi_2 \quad (4.23)$$

$$u.(\varphi_1 \vee \varphi_2) = u.\varphi_1 \vee u.\varphi_2 \quad (4.24)$$

$$[x : \pi_1].\varphi_1 \wedge [y : \pi_2].\varphi_2 = [x, y : \pi_1, \pi_2].\varphi_1 \wedge \varphi_2 \quad (4.25)$$
$$\text{si x différent de y et } \mathcal{F}(\varphi_1) \cap \mathcal{F}(\varphi_2) = \emptyset$$

$$(\psi_1\overline{\mathcal{U}}\varphi_2) \wedge (\psi_2\overline{\mathcal{U}}\varphi_2) = (\psi_1 \wedge \psi_2)\overline{\mathcal{U}}(\varphi_1 \wedge \varphi_2) \vee (\psi_1 \wedge \psi_2)\overline{\mathcal{U}}(\varphi_1 \wedge (\psi_2\overline{\mathcal{U}}\varphi_2)) \vee (\psi_1 \wedge \psi_2)\overline{\mathcal{U}}(\varphi_2 \wedge (\psi_1\overline{\mathcal{U}}\varphi_1)) \quad (4.26)$$

$$\bigcirc(\varphi_1 \vee \varphi_2) = \bigcirc\varphi_1 \vee \bigcirc\varphi_2 \quad (4.27)$$

$$\bigcirc\widetilde{\exists} x.\varphi = \widetilde{\exists} x.\bigcirc\varphi \quad (4.28)$$

$$\bigcirc[x : \pi].\varphi = (\pi \wedge [x : \pi].\bigcirc\varphi[x + 1/x]) \vee (\neg\pi \wedge [x : \pi].\bigcirc\varphi) \quad (4.29)$$

$$\bigcirc\varphi = [x : true].true\overline{\mathcal{U}}(x = 1 \wedge \bigcirc\varphi) \quad (4.30)$$

$$\psi\mathcal{U}(\varphi_1 \vee \varphi_2) = (\psi\mathcal{U}\varphi_1) \vee (\psi\mathcal{U}\varphi_2) \quad (4.31)$$

$$\psi\mathcal{U}(\widetilde{\exists} x.\varphi) = \widetilde{\exists} x.(\psi\mathcal{U}\varphi) \quad (4.32)$$

$$\psi\mathcal{U}\varphi = \varphi \vee (\psi\overline{\mathcal{U}}(\bigcirc\varphi)) \quad (4.33)$$

$$\psi\overline{\mathcal{U}}(\varphi_1 \vee \varphi_2) = (\psi\overline{\mathcal{U}}\varphi_1) \vee (\psi\overline{\mathcal{U}}\varphi_2) \quad (4.34)$$

$$\psi\overline{\mathcal{U}}(\widetilde{\exists} x.\varphi) = \widetilde{\exists} x.(\psi\overline{\mathcal{U}}\varphi) \quad (4.35)$$

$$\psi\overline{\mathcal{U}}\varphi = (\psi \wedge \varphi) \vee (\psi\overline{\mathcal{U}}(\bigcirc(\psi \wedge \varphi))) \quad (4.36)$$

Maintenant, nous obtenons la forme normale en remplaçant chaque contrainte d'occurrences $[\vec{x} : \vec{\pi}]$ par $[x_i : \pi_i]_{i=1}^n$ et en modifiant les contraintes d'occurrences en conséquence (un x qui était lié à π est remplacé dans une formule f par une somme de tout les x_j, tel

tel-00004890, version 1 - 19 Feb 2004

que π implique π_j). □

Exemple 4.1 :

Soient $\mathcal{P} = \{P, Q\}$ et $\Sigma = \{\{\}, \{P\}, \{Q\}, \{P, Q\}\}$. Alors $\pi_1 = \neg P \wedge \neg Q$, $\pi_2 = P \wedge \neg Q$, $\pi_3 = \neg P \wedge Q$ et $\pi_4 = P \wedge Q$. Pour transformer

$$\varphi = u.[x, y : P, Q].\ (P\overline{\mathcal{U}}(A^u \wedge (x \leq y)))$$

en forme normale, nous utilisons d'abord les règles (4.20) et (4.21) pour transformer A^u et $x \leq y$ en forme normale. Ensuite nous utilisons essentiellement la règle (4.26) pour obtenir une formule équivalente en forme normale. $[x, y : P, Q]$ est remplacée par $[x_i : \pi_i]_{i=1}^4$ et $x \leq y$ par $x_2 + x_4 \leq x_1 + x_3$.

Le corollaire suivant s'obtient en écrivant d'une autre manière les formules en forme normale.

Corollaire 4.3 :

Pour chaque formule de CLTL$_\diamond$ il y a une formule équivalente de CLTL$_\diamond$ qui est une disjonction de formules de la forme:

$$\widetilde{\exists}\vec{y}.\, u_0.\, [x_i^0 : \pi_i]_{i=1}^n.\, (\psi_0 \wedge (\psi_0' \,\overline{\mathcal{U}}\, (f_0 \wedge \bigcirc u_1.\, [x_i^1 : \pi_i]_{i=1}^n.\, (\psi_1 \wedge (\psi_1' \,\overline{\mathcal{U}}\, (f_1 \wedge \cdots \bigcirc u_m.\, [x_i^m : \pi_i]_{i=1}^n.\, (\psi_m \wedge (\psi_m' \,\overline{\mathcal{U}}\, (f_m \wedge \bigcirc u_{m+1}.\, \psi_{m+1}))) \cdots)))))) \quad (4.37)$$

4.5.2 Forme normale mono-contrainte

Nous définissons une deuxième forme normale pour CLTL$_\diamond$, appelée *mono-contrainte* et nous montrons que pour chaque formule de CLTL$_\diamond$ il existe une formule équivalente dans cette forme normale modulo projection. Les formules dans la forme normale mono-contrainte contiennent une seule contrainte arithmétique.

Définition 4.25 (Forme normale mono-contrainte) :

Soit φ une formule de CLTL$_\diamond$ de la forme (4.37). Soit at_j pour chaque $j \in \{0, \cdots, m+1\}$, une nouvelle proposition atomique. Soit $\lambda_j = \mathrm{at}_j \wedge \bigwedge_{k \neq j} \neg \mathrm{at}_k$. Soient $\mathcal{P}' = \mathcal{P} \cup \bigcup_{j=0}^{m+1} \{\mathrm{at}_j\}$ et $\Sigma' = 2^{\mathcal{P}'}$. Alors, la *forme normale mono-contrainte* de φ, appelé $\widehat{\varphi}$ est donné par:

$$\widetilde{\exists}\vec{y}.\, u_0.\, [z_i^j : \pi_i \wedge \lambda_j]_{i=1..n}^{j=0..m}.\, (\psi_0 \wedge ((\psi_0' \wedge \lambda_0)\,\overline{\mathcal{U}} \cdots \bigcirc u_m.\, (\psi_m \wedge ((\psi_m' \wedge \lambda_m)\,\overline{\mathcal{U}}\, ((\bigwedge_{j=0}^{m} g_j) \wedge \bigcirc u_{m+1}.\, (\psi_{m+1} \wedge \Box\lambda_{m+1})))) \cdots)) \quad (4.38)$$

tel-00004890, version 1 - 19 Feb 2004

où

$$g_j = f_j[\sum_{\ell=k}^{j} z_i^\ell / x_i^k]_{i=1..n}^{k=0..j}) \tag{4.39}$$

et les automates A dans la formule φ sont remplacés par des automates A' tel que $L(A') = \widetilde{L(A)}$

Proposition 4.4 :

Soit φ une formule de CLTL$_\diamond$ de la forme (4.37) et $\widehat{\varphi}$ sa forme normale mono-contrainte correspondante. Alors, $[\![\varphi]\!] = [\![\widehat{\varphi}]\!]|_\Sigma$.

Preuve:
Les λ_j sont mutuellement exclusive et à chaque variable z_i^j est associé $\pi_i \wedge \lambda_j$. Chaque variable z_i^j compte donc le nombre d'occurrences de π_i exactement dans la sous-séquence entre u_j (inclue) et u_{j+1} (exclue). Chaque contrainte de comptage f_j peut donc être déplacée vers la position u_{m+1} en remplaçant chaque variable de comptage x_i^k dans f_j, pour $k \leq j$ par la somme $\Sigma_{\ell=k}^{j} z_i^\ell$.
□

Exemple 4.2 :

Soit

$$\begin{aligned} \varphi \quad = \quad & u_0.[x_i^0 : \pi_i]_{i=1}^4.(P \wedge (Q\,\overline{\mathcal{U}}(x_1^0 > x_2^0 \wedge \\ & \bigcirc u_1.[x_i^1 : \pi_i]_{i=1}^4.(Q \wedge (P\,\overline{\mathcal{U}}(x_1^1 = x_2^0 \wedge \bigcirc u_2.true)))))) \end{aligned}$$

Alors

$$\begin{aligned} \widehat{\varphi} \quad = \quad & u_0.[z_i^0 : \pi_i \wedge \lambda_0]_{i=1}^4.[z_i^1 : \pi_i \wedge \lambda_1]_{i=1}^4.(P \wedge ((Q \wedge \lambda_0)\,\overline{\mathcal{U}} \\ & \bigcirc u_1.(Q \wedge ((P \wedge \lambda_1)\,\overline{\mathcal{U}}((z_1^0 > z_2^0 \wedge z_1^1 = z_2^0 + z_2^1) \wedge \\ & \quad \bigcirc u_2.(true \wedge \Box\lambda_2)))))) \end{aligned}$$

4.5.3 Décomposition

Nous définissons ici la décomposition d'une formule de CLTL$_\diamond$ en une formule de ALTL et une propriété d'inévitabilité de comptage.

Définition 4.26 :

Soit φ une formule de CLTL$_\diamond$ de la forme (4.37) et $\widehat{\varphi}$ sa forme normale mono-contrainte correspondante. Alors, soit $\widehat{\varphi}^*$ la formule de ALTL:

$$\begin{aligned} & u_0.\,(\psi_0 \wedge ((\psi_0' \wedge \lambda_0)\,\overline{\mathcal{U}} \\ & \quad \cdots \bigcirc u_m.\,(\psi_m \wedge ((\psi_m' \wedge \lambda_m)\,\overline{\mathcal{U}} \bigcirc u_{m+1}.\,(\psi_{m+1} \wedge \Box\lambda_{m+1}))) \cdots)) \end{aligned} \tag{4.40}$$

tel-00004890, version 1 - 19 Feb 2004

et $\widehat{\varphi}^{\#}$ la formule d'inévitabilité

$$[z_i^j : \pi_i \wedge \lambda_j]_{i=1..n}^{j=0..m} . [z : \lambda_{m+1}]. \diamond(z = 1 \wedge \exists \vec{y}. \bigwedge_{j=0}^{m} g_j) \qquad (4.41)$$

La formule

$$f_{\widehat{\varphi}} = (z = 1 \wedge \exists \vec{y}. \bigwedge_{j=0}^{m} g_j) \qquad (4.42)$$

est appelée *contrainte caractéristique* de φ.

Remarque: La quantification globale $\widetilde{\exists} \vec{y}$ dans (4.38) est remplacé dans (4.41) par une quantification $\exists \vec{y}$ de Presburger.

Exemple 4.3 :

Étant donné la formule φ de l'exemple 4.2 nous avons

$$\widehat{\varphi}^{*} = u_0.(P \wedge ((Q \wedge \lambda_0)\, \overline{\mathcal{U}} \bigcirc u_1.(Q \wedge ((P \wedge \lambda_1)\, \overline{\mathcal{U}} \bigcirc u_2.(true \wedge \Box \lambda_2))))))$$

et

$$\widehat{\varphi}^{\#} = [z_i^0 : \pi_i \wedge \lambda_0]_{i=1}^{n}.[z_i^1 : \pi_i \wedge \lambda_1]_{i=1}^{n}.[z : \lambda_2].\diamond(z = 1 \wedge (z_1^0 > z_2^0 \wedge z_1^1 = z_2^0 + z_2^1))$$

Proposition 4.5 (Décomposition) :

Soit φ une formule de CLTL$_\diamond$ de la forme (4.37) et $\widehat{\varphi}$ sa forme normale mono-contrainte correspondante. Alors $[\![\widehat{\varphi}]\!] = [\![\widehat{\varphi}^{*}]\!] \cap [\![\widehat{\varphi}^{\#}]\!]$.

Preuve:
Étant donnée une séquence qui satisfait $\widehat{\varphi}^{*}$, les positions $u_0, \ldots, u_{m+1}$ sont déterminées par la validité des λ_j's. La contrainte $z = 1$ assure en plus que les contraintes de comptages sont testées à u_{m+1}. □

Théorème 4.7 (Décomposition) :

Soit φ une formule fermée de CLTL$_\diamond$. Soit $\bigvee_{i=1}^{\ell} \varphi_i$ la formule équivalente en forme normale. Alors $[\![\varphi]\!] = \bigcup_{i=1}^{\ell}([\![\widehat{\varphi}_i^{*}]\!] \cap [\![\widehat{\varphi}_i^{\#}]\!])|_{\Sigma}$.

Preuve:
En utilisant les Propositions 4.5 et 4.4. □

tel-00004890, version 1 - 19 Feb 2004

4.5.4 Satisfaisabilité et validité relative de $\mathrm{CLTL}_{\Diamond}$ et $\mathrm{CLTL}_{\Box}$

Avec les résultats obtenus dans la section précédente nous pouvons montrer

Théorème 4.8 :

> Soient $S \subseteq \Sigma^\omega$, φ une formule fermée de $\mathrm{CLTL}_{\Diamond}$ et $\bigvee_{i=1}^{\ell} \varphi_i$ une formule en forme normale équivalente à φ. Alors, φ est satisfaisable relative à S si et seulement si $\bigcup_{i=1}^{\ell}(\widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!] \cap [\![\widehat{\varphi}_i^{\#}]\!]) \neq \emptyset$.

Preuve:
En utilisant le Théorème 4.7 et les Lemmes 2.1 et 2.4. □

Nous utilisons ce théorème pour obtenir des résultats généraux pour le problème de satisfaisabilite relative de $\mathrm{CLTL}_{Diamond}$ et de validité relative de $\mathrm{CLTL}_{\Box}$.

Théorème 4.9 :

> Soient $S \subseteq \Sigma^\omega$, φ une formule fermée de $\mathrm{CLTL}_{\Diamond}$ et $\bigvee_{i=1}^{\ell} \varphi_i$ une formule en forme normale équivalente à φ (d'après Proposition 4.3). Alors
>
> $$S \cap [\![\varphi]\!] \neq \emptyset \text{ ssi } \exists i \in \{1, \ldots, \ell\}.\ [Pref(\widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!])] \cap \langle\!| f_{\widehat{\varphi}_i} |\!\rangle \neq \emptyset. \tag{4.43}$$

Preuve:
D'après le Théorème 4.8, nous avons

$$S \cap [\![\varphi]\!] \neq \emptyset \text{ ssi } \exists i \in \{1, \ldots, \ell\}, \widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!] \cap [\![\widehat{\varphi}_i^{\#}]\!] \neq \emptyset$$

i.e., $\widehat{\varphi}_i^{\#}$ est satisfaisable relative à $\widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!]$. Alors, $\widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!] \cap [\![\widehat{\varphi}_i^{\#}]\!] \neq \emptyset$ ssi il existe un préfixe fini d'une séquence σ dans $\widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!]$ dont l'image de Parikh satisfait $f_{\widehat{\varphi}_i}$, c'est-à-dire pour un préfixe $\sigma(0, j)$ nous avons $[\sigma(0, j)] \in \langle\!| f_{\widehat{\varphi}_i} |\!\rangle$. □

Ce théorème permet de réduire le problème de la satisfaisabilité relative de formules $\mathrm{CLTL}_{\Diamond}$ au problème du vide d'ensemble semilinéaire, si l'ensemble S et la formule φ sont tels tous les $(\widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!])$'s sont semilinéaires.

Nous obtenons les trois théorèmes suivants:

Théorème 4.10 :

> Soit $\mathcal{C}$ une classe de ω-langages sur Σ tel que: Pour chaque $\Sigma' \supseteq \Sigma$, pour chaque $S \in \mathcal{C}$ et pour chaque langage ω-régulier R, $\widetilde{S} \cap R$ est semilinéaire. Alors, le problème de satisfaisabilité de formules fermées de $\mathrm{CLTL}_{\Diamond}$ relative aux ω-langages de $\mathcal{C}$ est décidable et par conséquent le problème de validité de $\mathrm{CLTL}_{\Box}$ relative à $\mathcal{C}$ est décidable.

tel-00004890, version 1 - 19 Feb 2004

Preuve:
En utilisant le Théorème 4.9 et le fait que toutes les formules $\widehat{\varphi}_i^*$ sont des formules de ALTL, i.e $[\![\widehat{\varphi}_i^*]\!]$ est un langage ω-régulier (d'après le Théorème 4.3). $\square$

Théorème 4.11 :

Soit $\mathcal{C}$ une classe de ω-langages sur Σ tel que: Pour chaque $\Sigma' \supseteq \Sigma$, pour chaque $S \in \mathcal{C}$ et pour chaque ω-langage sans hauteur d'étoile R, $\widetilde{S} \cap R$ est semilinéaire. Alors, le problème de satisfaisabilité de formules fermées de PLTL$_\diamond$ relative aux ω-langages de $\mathcal{C}$ est décidable et par conséquent le problème de validité de PLTL$_\square$ relative à $\mathcal{C}$ est décidable.

Preuve:
En utilisant le Théorème 4.9 et le fait que dans ce cas toutes les formules $\widehat{\varphi}_i^*$ sont des formules de LTL. $\square$

Théorème 4.12 :

Soit $\mathcal{C}$ une classe de ω-langages sur Σ tel que: Pour chaque $\Sigma' \supseteq \Sigma$, pour chaque $S \in \mathcal{C}$ et pour chaque langage ω-régulier simple R, $\widetilde{S} \cap R$ est semilinéaire. Alors, le problème de satisfaisabilité de formules fermées de PLTL$_\diamond$ simple relative aux ω-langages de $\mathcal{C}$ est décidable et par conséquent le problème de validité de PLTL$_\square$ simple relative à $\mathcal{C}$ est décidable.

Preuve:
En utilisant le Théorème 4.9 et le fait que dans ce cas chaque formule $[\![\widehat{\varphi}_i^*]\!]$ est un langage ω-régulier simple. $\square$

Théorème 4.10 nous permets de déduire plusieurs résultats de décidabilité. D'abord nous étudions le problème de *satisfaisabilité* et *validité* de CLTL$_\diamond$ respectivement CLTL$_\square$. En mettant $\mathcal{C} = \{\Sigma^\omega\}$ nous obtenons avec Théorème 4.10:

Corollaire 4.4 :

Le problème de satisfaisabilité de formules fermées de CLTL$_\diamond$ est décidable.

Corollaire 4.5 :

Le problème de validité de formules fermées de CLTL$_\square$ est décidable.

tel-00004890, version 1 - 19 Feb 2004

4.6 Le problème de la vérification pour $\mathrm{CLTL}_{\Box}$

Nous considérons dans cette section le problème de la *vérification* de systèmes. Nous commençons avec les ω-automates à pile.

4.6.1 Vérification des ω-automates à pile

Théorème 4.13 ($\mathrm{CLTL}_{\Box}$ vs. ω-automates à pile) :

> Le problème de satisfaisabilité (resp. validité) de formules de $\mathrm{CLTL}_{\Diamond}$ (resp. $\mathrm{CLTL}_{\Box}$) relative à des langages ω-hors-contexte est décidable. En particulier, le problème de la vérification de ω-automate à pile par rapport à des formules fermées de $\mathrm{CLTL}_{\Box}$ est décidable.

Preuve:
En utilisant le Théorème 4.10 et

- Les langages ω-hors-contextes sont fermés par cylindrification: Soit L un langage ω hors-contexte. Dans le ω-automate à pile qui définit L nous remplaçons dans la relation de transition chaque $\sigma \in \Sigma^*$ par toutes les $\sigma \in (\Sigma')^*$ avec $|\sigma'|_{\Sigma} = \sigma$.
- Les langages ω-hors-contextes sont fermés par intersection avec des langages ω-réguliers (Théorème 2.3),
- Les langages ω-hors-contextes sont semilinéaires (Théorème 2.8).

□

Corollaire 4.6 :

> Le problème de satisfaisabilité (resp. validité) de formules de $\mathrm{CLTL}_{\Diamond}$ (resp. $\mathrm{CLTL}_{\Box}$) relative à des ω-langages BPA et des ω-langages d'automate à pile est décidable. En particulier, le problème de la vérification des processus BPA et des automates à pile par rapport à des formules fermées de $\mathrm{CLTL}_{\Box}$ est décidable.

Pour les processus BPP nous avons montré que l'intersection avec un ω-langage sans hauteur d'étoile peut donner un langage qui n'est pas semilinéaire (voir la Proposition 2.2). Par conséquent nous ne pouvons pas appliquer l'approche précédente pour résoudre le problème de la vérification. Ce problème est néanmoins décidable (voir section 4.6.3).

4.6.2 Vérification de processus PA

Dans cette section nous considérons le problème de la vérification de processus PA.

tel-00004890, version 1 - 19 Feb 2004

4.6.2.1 Résultat d'indécidabilité

Nous pouvons montrer que le Théorème 4.13 n'est pas vrai pour toute la classe des systèmes semilinéaires. En effet, nous avons le théorème suivant:

Théorème 4.14 (LTL vs. processus PA) :

> Le problème de satisfaisabilité de formules de LTL relative à des ω-langages semilinéaires (en particulier les PA ω-langages) est indécidable. Par conséquent, le problème de la vérification de systèmes semilinéaire par rapport à LTL est indécidable.

Preuve:
Nous pouvons coder le problème de l'arrêt d'une machine à deux compteurs (voir section 4.3) comme le fait que l'intersection d'un ω-langage sans hauteur d'étoile R avec un ω-langage S n'est pas vide. Le langage S est donné par un processus PA $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ construit comme suit: $\mathcal{S}$ est donné par la composition parallèle (sans synchronisation) de deux processus BPA qui simulent les deux compteurs et un processus régulier qui simule le contrôle fini de la machine. L'intersection avec un langage très simple R donne ensuite toutes les calculs "correctes" de la machine.

Soit $\mathcal{M}$ une machine à deux compteur avec m instructions. Nous construisons d'abord pour chaque compteur de la machine un processus BPA qui le simule.
Soit $\mathcal{S}^i = (Var^i, \Sigma^i, \Delta^i, X_1^i)$ pour $i \in \{1, 2\}$ avec

- $Var^i = \{X_1^i, X_2^i\}$,
- $\Sigma^i = \{inc_i, dec_i, z_i\}$,
- Les éléments de Δ^i sont donnés par
  - $X_1^i \mathrel{\widehat{=}} z_i \cdot X_1^i$,
  - $X_1^i \mathrel{\widehat{=}} inc_i \cdot X_2^i \cdot X_1^i$,
  - $X_2^i \mathrel{\widehat{=}} dec_i$,
  - $X_2^i \mathrel{\widehat{=}} inc_i \cdot X_2^i$.

Intuitivement, le terme X_1^i représente la valeur zéro et le terme $X_2^i \cdots X_2^i \cdot X_1^i$ avec n X_2^i's représente la valeur n du compteur i.

Nous simulons le contrôle fini de la machine par un processus régulier $\mathcal{C} = (Var^c, \Sigma^c, \Delta^c, X_1^c)$ donné par:

- $Var^c = \{X_i^c \ : 1 \leq i \leq m\}$,
- $\Sigma^c = \{inc_1^c, dec_1^c, z_1^c, inc_2^c, dec_2^c, z_2^c, halt\}$,
- Δ^c est le plus petit ensemble qui satisfait:
  - Si $(s_j : c_i := c_i{+}1;\ goto\ s_k)$ est une commande de M, alors $(X_j \mathrel{\widehat{=}} inc_i^c \cdot X_k) \in \Delta^c$,

- Si (s_j : *if* $c_i = 0$ *then goto* s_{k_1} *else* $c_i := c_i \Leftrightarrow 1$ *goto* s_{k_2}) est une commande de $\mathcal{M}$, alors $(X_j \mathrel{\widehat{=}} z_i^c \cdot X_{k_1}) \in \Delta^c$ et $(X_j \mathrel{\widehat{=}} dec_i^c \cdot X_{k_2}) \in \Delta^c$
- $X_m \mathrel{\widehat{=}} halt \cdot X_m \in \Delta^c$

Nous ajoutons $X_m \mathrel{\widehat{=}} halt \cdot X_m$ pour avoir des séquences infinies.

Maintenant, le processus PA $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ est donné par:

- $Var = Var^1 \cup Var^2 \cup Var^c \cup \{X_1\}$,
- $\Sigma = \Sigma^1 \cup \Sigma^2 \cup \Sigma^c$
- $\Delta = \Delta^1 \cup \Delta^2 \cup \Delta^c \cup \{X_1 \mathrel{\widehat{=}} X_1^1 \| X_1^2 \| X_1^c\}$,

Soit R le ω-langage sans hauteur d'étoile donné par l'expression ω-régulière

$$((inc_1.inc_1^c) + (dec_1.dec_1^c) + (z_1.z_1^c) + (inc_2.inc_2^c) + (dec_2.dec_2^c) + (z_2.z_2^c))^*.halt^\omega$$

Alors

$$R \cap L(\mathcal{S}) \neq \emptyset \text{ ssi M s'arrête}$$

puisque R force la synchronisation entre les deux compteurs et le contrôle fini. □

4.6.2.2 Résultat de décidabilité

Si nous nous restreignons aux propriétés de $\mathrm{PLTL}_\Diamond$ simple nous pouvons néanmoins montrer la décidabilité du problème de satisfaisabilité relative à des ω-langages PA.

Théorème 4.15 ($\mathrm{PLTL}_\Box$ simple vs. processus PA) :

> Le problème de satisfaisabilité (resp. validité) de formules fermées de $\mathrm{PLTL}_\Diamond$ simple (resp. $\mathrm{PLTL}_\Box$ simple) relative à des ω-langages PA est décidable. Par conséquent, le problème de la vérification de processus PA par rapport à des formules fermées de $\mathrm{PLTL}_\Box$ simple est décidable.

Preuve:

Nous utilisons les notations de la section 2.5.4. Soient $\mathcal{S} = (Var, \Sigma, \Delta, X_1)$ un processus PA et φ une formule de $\mathrm{PLTL}_\Diamond$ simple. Soit S le ω-langage de $\mathcal{S}$. Soit $\bigvee_{i=1}^{\ell} \varphi_i$ une formule en forme normale équivalente à φ. D'après le Théorème 4.9 nous avons: φ est satisfaisable relative à S si et seulement si

$$\exists i \in \{1, \ldots, \ell\}.\ [Pref(\widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!])] \cap \langle\!\langle f_{\widehat{\varphi}_i} \rangle\!\rangle \neq \emptyset$$

tel-00004890, version 1 - 19 Feb 2004

Nous fixons un i et étudions le problème

$$[Pref(\widetilde{S} \cap [\![\widehat{\varphi}_i^*]\!])] \cap \langle\!| f_{\widehat{\varphi}_i} |\!\rangle \neq \emptyset$$

D'abord nous pouvons facilement obtenir un processus $\mathcal{S}'$ avec $L(\mathcal{S}') = \widetilde{S}$ (comme dans la preuve du théorème 4.13 pour les automates à pile). La formule $\widehat{\varphi}_i^*$ a la forme

$$(\psi_0 \wedge ((\psi_0' \wedge \lambda_0) \overline{\mathcal{U}} \cdots \bigcirc(\psi_m \wedge ((\psi_m' \wedge \lambda_m) \overline{\mathcal{U}} \bigcirc (\psi_{m+1} \wedge \Box\lambda_{m+1}))) \cdots))$$

Puisque φ est une formule de PLTL$_\Diamond$ simple, nous savons que $\psi_0, \ldots, \psi_m$ sont des *formules propositionnelles* et $\psi_0', \ldots, \psi_{m+1}'$ sont des formules composées de formules propositionnelles et de $\Box\psi$ (où ψ est une formule propositionnelle). Sans perte de généralité nous pouvons supposer que $\Box\psi$ n'apparaît que dans ψ_{m+1}' ($(\Box\psi_1)\overline{\mathcal{U}}\psi_2$ est équivalente à $\psi_1\overline{\mathcal{U}}(\psi_2 \wedge \Box\psi_1)$). Par conséquent, $L = [\![\widehat{\varphi}_i^*]\!]$ peut être écrit de la forme:

$$B_{-1,0}.B_{0,0}^*.B_{0,1}.B_{1,1}^*.B_{1,2} \ldots B_{m,m}^*.B_{m,m+1}.B_{m+1,m+1}^\omega$$

avec toutes les $B_{i,i} \subseteq \Sigma'$ et $B_{i,i+1} \subseteq \Sigma'$.

Puisque $f_{\widehat{\varphi}_i} = (z = 1 \wedge \exists \vec{y}. \bigwedge_{j=0}^{m} g_j)$ et z compte les λ_{m+1}, il suffit de considérer dans $Pref(L(\mathcal{S}') \cap L)$ les préfixes qui contiennent exactement un $b \in \Sigma'$ tel que $b \models \lambda_{m+1}$. Soit P' cet ensemble. La construction de $\widehat{\varphi}_i^*$ implique que tous les éléments de $B_{m,m+1}$ satisfont λ_{m+1} et tous les éléments des $B_{i,i}$ et $B_{i,i+1}$ avant ne satisfont pas λ_{m+1}. Il suffit donc de montrer le lemme suivant:

Lemme 4.2 :

> $[P']$ est semilinéaire.

Nous donnons d'abord une idée intuitive de la preuve. Nous devons caractériser l'ensemble $[P']$. Il est facile de voir qu'un élément σ' de P' est donné par une séquence de la forme

$$X_1 \overset{b_{-1,0}}{\Leftrightarrow\!\rightarrow}_{\mathcal{S}'} \tau_0 \overset{\sigma_{0,0}}{\Leftrightarrow\!\rightarrow}_{\mathcal{S}'} \tau_{0,1} \overset{b_{0,1}}{\Leftrightarrow\!\rightarrow}_{\mathcal{S}'} \tau_{1,1} \cdots \tau_{m,m+1} \overset{b_{m,m+1}}{\Leftrightarrow\!\rightarrow}_{\mathcal{S}'} \tau_{m+1,m+1} \overset{\sigma_{m+1,m+1}}{\Leftrightarrow\!\rightarrow}_{\mathcal{S}'} \tau_{m+1,m+2} \cdots \tag{4.44}$$

où

- $\sigma' = b_{-1,0}\sigma_{0,0}b_{0,1}\sigma_{1,1} \cdots b_{m,m+1}$
- $\forall i \in \{\Leftrightarrow 1, \ldots, m\}.\ b_{i,i+1} \in B_{i,i+1}$
- $\forall i \in \{0, \ldots, m+1\}.\ \sigma_{i,i} \in (\Sigma')^*$ avec $\forall j \in \{0, \ldots, |\sigma_{i,i}| \Leftrightarrow 1\}.\ \sigma_{i,i}(j) \in B_{i,i}$
- Le terme $\tau_{m+1,m+2}$ contient une variable exécutable qui est dans l'ensemble $Boucle_{\mathcal{S}'}^{\pi}$ (voir la définition 2.60), où $\pi = \bigvee_{b \in B_{m+1,m+1}} (\bigwedge_{P \in b} P \wedge \bigwedge_{P \notin b} \neg P)$.

tel-00004890, version 1 - 19 Feb 2004

Puisque nous sommes seulement intéressés par l'image de Parikh de P' l'ordre dans lequel les éléments de la séquence σ' sont produits n'est pas important. Nous construisons une grammaire hors-contexte $\mathcal{G} = (Var_{\mathcal{G}}, \Sigma', R, S)$ tel que $[L(\mathcal{G})] = [P']$. $\mathcal{G}$ est une grammaire qui produit les $\sigma' \in P'$ dans un autre ordre que $\mathcal{S}'$.

L'idée derrière la construction de cette grammaire est la suivante: Les symboles non-terminaux de la grammaire sont données par les variables de $\mathcal{S}'$ avec un ensemble d'indices qui indiquent à quel moment dans la séquence de la forme (4.44) la variable *et* les variables générées par elle "produisent" des symboles. Par exemple une variable de la forme $[X, \{(0,1),(2,2),(m+1,m+1),(m+2,m+2)\}]$ est une variable qui produit des symboles aux niveaux $(0,1),(2,2),(m+1,m+1)$ et $(m+2,m+2)$. Le niveau $(m+2,m+2)$ indique qu'un descendant de la variable apparaît dans le terme $\tau_{m+1,m+2}$. Dans les niveaux $(i,i+1)$ (pour $\Leftrightarrow 1 \leq i \leq m$) il y a exactement un symbole qui est produit. La variable à qui est associée le niveau $(m+1,m+2)$ doit être une variable dans l'ensemble $Boucle^{\pi}_{\mathcal{S}'}$.

Chaque règle du processus PA est traduit vers plusieurs productions de la grammaire hors-contexte qui indiquent toutes les possibilités de découper l'ensemble d'indice associé à la variable. Ce découpage doit être compatible avec les opérateurs $\|$ et $\cdot$ dans les règles.

Par exemple, soit X une variable et $\{(0,1),(1,1),(3,3)\}$ un ensemble d'indice. L'ensemble $\{(0,1),(1,1),(3,3)\}$ indique que X doit produire un symbole au niveau $(0,1)$ et aux niveaux $(1,1)$ et $(3,3)$. Nous avons donc besoin d'une règle dans $\mathcal{S}'$ pour X qui produit un symbole $a \in B_{0,1}$. Soit donnée une règle de la forme $X \mathrel{\widehat{=}} a \cdot (Y \| Z)$. Dans la grammaire il y aura les règles

$$
\begin{array}{lcl}
[X,\{(0,1),(1,1),(3,3)\}] & \rightarrow & a.[Y,\{(1,1),(3,3)\}].[Z,\{(1,1),(3,3)\}] \\
[X,\{(0,1),(1,1),(3,3)\}] & \rightarrow & a.[Y,\{(1,1),(3,3)\}].[Z,\{(1,1)\}] \\
[X,\{(0,1),(1,1),(3,3)\}] & \rightarrow & a.[Y,\{(1,1),(3,3)\}].[Z,\{(3,3)\}] \\
[X,\{(0,1),(1,1),(3,3)\}] & \rightarrow & a.[Y,\{(1,1)\}].[Z,\{(1,1),(3,3)\}] \\
[X,\{(0,1),(1,1),(3,3)\}] & \rightarrow & a.[Y,\{(1,1)\}].[Z,\{(3,3)\}] \\
[X,\{(0,1),(1,1),(3,3)\}] & \rightarrow & a.[Y,\{(3,3)\}].[Z,\{(1,1),(3,3)\}] \\
[X,\{(0,1),(1,1),(3,3)\}] & \rightarrow & a.[Y,\{(3,3)\}].[Z,\{(1,1)\}]
\end{array}
$$

Nous construisons la grammaire formellement comme suit: Nous pouvons d'abord supposer que $\mathcal{S}'$ est en forme normale simplifiée. Nous définissons ensuite l'ensemble de niveaux possibles et les fonctions qui permettent de répartir les indices sur les variables en respectant les opérateurs de composition parallèle et séquentielle.

Soient

$$Ind = \{(i,i+1) \ : \ \Leftrightarrow 1 \leq i \leq m+1\} \cup \{(i,i) \ : \ 0 \leq i \leq m+2\}$$

$$J = \{(i,i+1) \ : \ \Leftrightarrow 1 \leq i \leq m+1\}$$

et $I \subseteq Ind$. Soient *seq* et *par* deux fonctions définies comme suit:

$$
\begin{array}{lcl}
seq(I) := \{(I_1, I_2) & : & I_1 \cup I_2 = I \wedge I_1 \neq \emptyset \wedge I_2 \neq \emptyset \\
 & & \wedge((max(I_1) = min(I_2) \wedge max(I_1) \notin J) \vee I_1 \cap I_2 = \emptyset)
\end{array}
$$

tel-00004890, version 1 - 19 Feb 2004

et

$$par(I) := \{(I_1, I_2) \ : \ I_1 \cup I_2 = I \wedge I_1 \neq \emptyset \wedge I_2 \neq \emptyset \wedge I_1 \cap I_2 \cap J = \emptyset\}$$

Soit $Te(t, I)$ avec $t \in \mathcal{T}$ et $I \subseteq \{1, \ldots, m+1\}$ définie récursivement comme suit:

- $Te(\mathbf{0}, I) = \{\epsilon\}$,
- $Te(X, I) = \{[X, I]\}$,
- $Te(\tau_1 \cdot \tau_2, I) = \bigcup_{(I_1,I_2) \in seq(I)} \{\tau_1' \tau_2' \ : \ \tau_1' \in Te(\tau_1, I_1) \wedge \tau_2' \in Te(\tau_2, I_2)\}$,
- $Te(\tau_1 \| \tau_2, I) = \bigcup_{(I_1,I_2) \in par(I)} \{\tau_1' \tau_2' \ : \ \tau_1' \in Te(\tau_1, I_1) \wedge \tau_2' \in Te(\tau_2, I_2)\}$

Soit $\pi = \bigvee_{b \in B_{m+1,m+1}} (\bigwedge_{P \in b} P \wedge \bigwedge_{P \notin b} \neg P)$. $\mathcal{G} = (Var_{\mathcal{G}}, \Sigma', R, S)$ est donnée par:

- $Var_{\mathcal{G}} = \{[X, I] : X \in Var' \wedge I \subseteq Ind\}$.
- R est le plus petit ensemble qui satisfait:
  - si $X_i \mathrel{\widehat{=}} t_i \in \Delta'$ avec $t_i = a_1^i \cdot t_1^i + \cdots + a_{m_i}^i \cdot t_{m_i}^i$ alors
    $\forall I \subseteq Ind$ avec $min(I) \notin \{(m+1, m+1), (m+1, m+2), (m+2, m+2)\})$
    $\forall j \in \{1, \ldots, m_i\}$.
    $\forall t \in Te(t_j^i, I)$.
    $a_j^i \in B_{min(I)}$ et $min(I) \notin J$ implique
    $[X_i, I] \rightarrow_{\mathcal{G}} a_j^i.t$
    et $\forall t \in Te(t_j^i, I \setminus min(I))$
    $a_j^i \in B_{min(I)}$ implique
    $[X_i, I] \rightarrow_{\mathcal{G}} a_j^i.t$
    $\wedge \ \forall I \subseteq \{(m+1, m+1), (m+1, m+2), (m+2, m+2)\}$
    $\forall j \in \{1, \ldots, m_i\}$.
    $\forall t \in Te(t_j^i, I)$.
    $a_j^i \in B_{min(I)}$ et $min(I) \notin J$ implique
    $[X_i, I] \rightarrow_{\mathcal{G}} t$
    et $\forall t \in Te(t_j^i, I \setminus min(I))$
    $a_j^i \in B_{min(I)}$ implique
    $[X_i, I] \rightarrow_{\mathcal{G}} t$
  - $[X_i, \{(m+1, m+2), (m+2, m+2)\}] \rightarrow_{\mathcal{G}} \epsilon$ si $X_i \in Boucle_{\mathcal{S}'}^{\pi}$
  - $[X_i, \{(m+1, m+2)\}] \rightarrow_{\mathcal{G}} \epsilon$ si $X_i \in Boucle_{\mathcal{S}'}^{\pi}$
  - $[X_i, (m+2, m+2)] \rightarrow_{\mathcal{G}} \epsilon$
  - $\forall I \supseteq Ind$ avec $J \subseteq I \cap Ind$. $S \rightarrow_{\mathcal{G}} [X, I]$.

tel-00004890, version 1 - 19 Feb 2004

Nous prouvons dans la suite que pour chaque σ' pour lequel il existe une séquence de la forme (4.44) il existe un mot w produit par la grammaire hors-contexte qui a le même image de Parikh (c'est-à-dire $[\sigma'] = [w]$) et vice versa.

– Soit $\sigma' \in P'$ tel qu'il existe une séquence de la forme (4.44). Alors nous connaissons pour chaque variable qui est exécutée dans cette séquence le niveau dans lequel elle est exécutée. Et nous pouvons d'une manière évidente construire une dérivation d'un mot dans la grammaire qui a le même image de Parikh que σ'. Par exemple: Soient $m = 1$ et Y une variable qui permet de répéter continûment π. Considérons la séquence suivante d'un processus PA:

$$X \overset{\mathbf{b_{-1,0}}}{\Leftrightarrow} Y \| Z \overset{\mathbf{b_{0,1}}}{\Leftrightarrow} (X.Y) \| Z \overset{\mathbf{b_{1,1}}}{\Leftrightarrow} (X.Y) \| (Y.Z) \overset{\mathbf{b'_{1,1}}}{\Leftrightarrow} (X.Y.Y) \| (Y.Z)$$
$$\overset{\mathbf{b_{1,2}}}{\Leftrightarrow} (X.Y.Y) \| Z \overset{\mathbf{b_{2,2}}}{\Leftrightarrow} (Y.Y) \| Z$$

Cette séquence correspond à la dérivation dans la grammaire:

$$\begin{aligned}
X &\Rightarrow_{\mathcal{G}} [X, \{(-1,0),(0,1),(1,1),(1,2),(2,2),(2,3),(3,3)\}] \\
&\Rightarrow_{\mathcal{G}} \mathbf{b_{-1,0}}.[Y, \{(0,1),(1,1),(2,2),(2,3)\}].[Z, \{(1,1),(1,2),(3,3)\}] \\
&\Rightarrow_{\mathcal{G}} \mathbf{b_{-1,0}}.\mathbf{b_{0,1}}.[X, \{(1,1),(2,2),(2,3)\}].[Y, \{(3,3)\}].[Z, \{(1,1),(1,2),(3,3)\}] \\
&\Rightarrow_{\mathcal{G}} \mathbf{b_{-1,0}}.\mathbf{b_{0,1}}.[X, \{(1,1),(2,2),(2,3)\}].[Y, \{(3,3)\}].\mathbf{b_{1,1}}.[Y, \{(1,2)\}].[Z, \{(3,3)\}] \\
&\Rightarrow_{\mathcal{G}} \mathbf{b_{-1,0}}.\mathbf{b_{0,1}}.\mathbf{b'_{1,1}}.[X, \{(2,2)\}].[Y, \{(2,3)\}].[Y, \{(3,3)\}].\mathbf{b_{1,1}}.[Y, \{(1,2)\}].[Z, \{(3,3)\}] \\
&\Rightarrow_{\mathcal{G}} \mathbf{b_{-1,0}}.\mathbf{b_{0,1}}.\mathbf{b'_{1,1}}.[X, \{(2,2)\}].[Y, \{(2,3)\}].[Y, \{(3,3)\}].\mathbf{b_{1,1}}.\mathbf{b_{1,2}}.[Z, \{(3,3)\}] \\
&\Rightarrow_{\mathcal{G}} \mathbf{b_{-1,0}}.\mathbf{b_{0,1}}.\mathbf{b'_{1,1}}.[Y, \{(2,3)\}].[Y, \{(3,3)\}].\mathbf{b_{1,1}}.\mathbf{b_{1,2}}.[Z, \{(3,3)\}] \\
&\Rightarrow_{\mathcal{G}} \mathbf{b_{-1,0}}.\mathbf{b_{0,1}}.\mathbf{b'_{1,1}}.\mathbf{b_{1,1}}.\mathbf{b_{1,2}}
\end{aligned}$$

– Soit w un mot produit par la grammaire. Alors, il existe une dérivation $S \Rightarrow^*_{\mathcal{G}} w$. Par construction de la grammaire nous savons que pour chaque indice $(i, i+1) \in J$ cette dérivation contient au maximum une application d'une règle de la forme $[X, I] \rightarrow_{\mathcal{G}} t$ où $min(I) = (i, i+1)$. Nous pouvons supposer que dans cette dérivation l'application des règles de la grammaire est ordonnée par niveaux: D'abord la règle avec la partie de gauche de la forme $[X, I]$ avec $min(I) = (\Leftrightarrow 1, 0)$, ensuite les règles qui ont une partie gauche de la forme $[X, I]$ avec $min(I) = (0, 0)$ (il n'y en a peut-être pas), après la règle pour $[X, I]$ avec $min(I) = (0, 1)$, et ainsi de suite. À partir de cette dérivation nous pouvons facilement construire une suite de la forme (4.44) telle que $\sigma' \in P'$ et $[\sigma'] = [w]$ en choisissant pour chaque application d'une règle de la grammaire la règle correspondante dans le processus $\mathcal{S}'$. Les productions de la forme $[X_i, \{(m+1, m+2), (m+2, m+2)\}] \rightarrow_{\mathcal{G}} \epsilon$, $[X_i, \{(m+1, m+2)\}] \rightarrow_{\mathcal{G}} \epsilon$ et $[X_i, (m+2, m+2)] \rightarrow_{\mathcal{G}} \epsilon$ ne correspondent pas à l'application d'une règle du processus $\mathcal{S}'$ mais signifient que la variable de la partie gauche appartient au terme $\tau_{m+1,m+2}$.

□

tel-00004890, version 1 - 19 Feb 2004

4.6.3 Vérification de réseaux de Petri

Nous étudions dans cette section le problème de satisfaisabilité de formules fermées de CLTL$_\Diamond$ relative aux réseaux de Petri. Ces systèmes ne sont pas semilinéaires. Nous ne pouvons donc pas appliquer le même raisonnement que dans la section précédente et réduire le problème au problème du vide des ensembles semilinéaires. Néanmoins nous pouvons montrer la décidabilité du problème en le transformant au problème d'atteignabilité des réseaux de Petri. Ce résultat étend le résultat de Esparza [Esp94] pour les réseaux de Petri et le μ-calcul linéaire, parce que CLTL$_\Box$ est strictement plus expressif que le μ-calcul linéaire. En plus, CLTL$_\Box$ permet d'exprimer des contraintes sur les places d'un réseaux de Petri en comptant les transitions entrantes et sortantes. Alors, le résultat peut être considéré comme un résultat de décidabilité pour une logique sur les marquages de réseaux de Petri. Dans ce contexte, notre résultat est incomparable aux résultats existants [HRY91, Jan90].

Soient $\mathcal{N}$ un réseau de Petri, φ une formule fermée de CLTL$_\Diamond$ et $\bigvee_{i=1}^{\ell} \varphi_i$ la formule en forme normale équivalente à φ. En utilisant le Théorème 4.8, nous avons

$$L(\mathcal{N}) \cap [\![\varphi]\!] \neq \emptyset \text{ ssi } \exists i \in \{1, \ldots, \ell\}, \widetilde{L(\mathcal{N})} \cap [\![\widehat{\varphi}_i^*]\!] \cap [\![\widehat{\varphi}_i^\#]\!] \neq \emptyset$$

où $\widetilde{L(\mathcal{N})}$ et les $\widehat{\varphi}_i^*$'s et $\widehat{\varphi}_i^\#$ sont définis sur le nouvel alphabet Σ'. Fixons un $i \in \{1, \ldots, \ell\}$ et étudions le problème

$$\widetilde{L(\mathcal{N})} \cap [\![\widehat{\phi}_i^*]\!] \cap [\![\widehat{\phi}_i^\#]\!] \neq \emptyset$$

Le réseau $\mathcal{N}$ peut être transformé facilement dans un réseau $\widetilde{\mathcal{N}}$ sur Σ' tel que $L(\widetilde{\mathcal{N}}) = \widetilde{L(\mathcal{N})}$. Par définition de $[\![\widehat{\varphi}_i^\#]\!]$ nous avons

$$\widetilde{L(\mathcal{N})} \cap [\![\widehat{\varphi}_i^*]\!] \cap [\![\widehat{\varphi}_i^\#]\!] \neq \emptyset \text{ ssi } \exists \sigma \in Pref(L(\widetilde{\mathcal{N}}) \cap [\![\widehat{\varphi}_i^*]\!]).\ [\sigma] \models f_{\widehat{\varphi}_i} \quad (4.45)$$

$\widehat{\varphi}_i^*$ est une formule de ALTL et nous pouvons construire un automate de Büchi $\mathcal{A} = (\mathcal{S}_\mathcal{A}, F)$ tel que $L(\mathcal{A}) = [\![\widehat{\varphi}_i^*]\!]$.

Soit $\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}$ le réseau produit de $\widetilde{\mathcal{N}}$ et $\mathcal{S}_\mathcal{A}$ en considérant chaque état de $\mathcal{S}_\mathcal{A}$ comme une place. Chaque transition de $\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}$ a comme source une transition de $\widetilde{\mathcal{N}}$ et de $\mathcal{S}_\mathcal{A}$. Soit $\mathbf{T}_\infty$ l'ensemble de transitions de $\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}$ qui a comme source une transition de $\mathcal{S}_\mathcal{A}$ de la forme $q \to q'$, avec $q \in F$. Alors, $L(\widetilde{\mathcal{N}}) \cap [\![\widehat{\phi}_i^*]\!]$ est l'ensemble de séquences infinies sur Σ' générées par des séquences de transitions dans $\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}$ qui contiennent infiniment souvent une des transitions de $\mathbf{T}_\infty$.

$Pref(L(\widetilde{\mathcal{N}}) \cap [\![\widehat{\phi}_i^*]\!])$ peut donc être caractérisé comme l'ensemble de séquences finies sur Σ' générées par des séquences de transitions de $\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}$ qui atteignent des marquages à partir desquels il y a des séquences infinies de transitions avec infiniment souvent une transition de $\mathbf{T}_\infty$.

L'ensemble de ces marquages est semilinéaire et peut être construit en utilisant le résultat suivant prouvé dans [VJ85]:

tel-00004890, version 1 - 19 Feb 2004

Lemme 4.3 (Valk et Jantzen) :

Soient $\mathcal{N}$ un réseau de Petri et t une des transitions. Alors, l'ensemble $\mathcal{M}_\infty(\mathcal{N}, t)$ (voir définition 2.65) est semilinéaire et peut être construit.

Soit $\mathcal{L}_\infty$ l'ensemble semilinéaire

$$\bigcup_{t \in \mathbf{T}_\infty} \mathcal{M}_\infty(\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}, t)$$

Alors, avec (4.45) nous obtenons

$$\widetilde{L(\mathcal{N})} \cap [\![\widehat{\varphi}_i^*]\!] \cap [\![\widehat{\varphi}_i^\#]\!] \neq \emptyset \ \text{ ssi } \ \exists \sigma \in (\Sigma')^*. \, \exists M \in \mathcal{L}_\infty. \, M_{\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}} \xrightarrow{\sigma} M \text{ et } [\sigma] \models f_{\widehat{\varphi}_i} \tag{4.46}$$

Pour traiter les contraintes de comptages nous étendons le réseau $\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}$ avec des nouvelles places qui codent les variables de comptages dans $f_{\widehat{\varphi}_i}$. À chaque nouvelle place est associée une étiquette de Σ'. La place à laquelle est associé par exemple a, compte le nombre de fois les transitions de $\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}$ étiqueté par a sont tirées. Soient $\mathbf{P}^\#$ l'ensemble de nouvelles places et $\widetilde{\mathcal{N}}_\mathcal{A}^\#$ le réseau étendu. $\langle\!|f_{\widehat{\varphi}_i}|\!\rangle$ est un ensemble semilinéaire inclus dans $I\!N^d$, si $d = |\mathbf{P}^\#|$. Soit $\langle\!|f_{\widehat{\varphi}_i}|\!\rangle'$ (resp. $\mathcal{L}'_\infty$) l'ensemble de tous les marquages de $\widetilde{\mathcal{N}}_\mathcal{A}^\#$ dont les projection sur $\mathbf{P}^\#$ (resp. les places de $\widetilde{\mathcal{N}} \times \mathcal{S}_\mathcal{A}$) sont dans $\langle\!|f_{\widehat{\varphi}_i}|\!\rangle$ (resp. $\mathcal{L}_\infty$). Les ensembles $\langle\!|f_{\widehat{\varphi}_i}|\!\rangle'$ et $\mathcal{L}'_\infty$ sont semilinéaires et peuvent être facilement construit à partir de $\langle\!|f_{\widehat{\varphi}_i}|\!\rangle$ et $\mathcal{L}_\infty$. Puisque la classe des ensembles semilinéaires est fermée par intersection, l'ensemble $\mathcal{L}'_\infty \cap \langle\!|f_{\widehat{\varphi}_i}|\!\rangle'$ est aussi semilinéaire. Il s'en suit avec (4.46) que:

$$\widetilde{L(\mathcal{N})} \cap [\![\widehat{\varphi}_i^*]\!] \cap [\![\widehat{\varphi}_i^\#]\!] \neq \emptyset \ \text{ ssi } \ \exists \sigma \in (\Sigma')^*. \, \exists M \in (\mathcal{L}'_\infty \cap \langle\!|f_{\widehat{\varphi}_i}|\!\rangle'). \, M_{\widetilde{\mathcal{N}}_\mathcal{A}^\#} \xrightarrow{\sigma} M \tag{4.47}$$

Nous avons par conséquent réduit le problème $\widetilde{L(\mathcal{N})} \cap [\![\widehat{\varphi}_i^*]\!] \cap [\![\widehat{\varphi}_i^\#]\!] \neq \emptyset$ vers un problème d'atteignabilité d'un ensemble semilinéaire d'un réseau de Petri, i.e. le problème s'il existe un marquage atteignable inclus dans un ensemble semilinéaire donné. En utilisant le Théorème 2.6 ce problème est décidable et nous obtenons le résultat:

Théorème 4.16 (CLTL$_\Box$ vs. réseaux de Petri) :

Le problème de satisfaisabilité de formules fermées de CLTL$_\Diamond$ relative au réseaux de Petri est décidable. Le problème de la vérification des réseaux de Petri par rapport à des formules fermées de CLTL$_\Box$ est par conséquent décidable.

4.7 Conclusion

Nous terminons ce chapitre avec un tableau récapitulatifs des résultats concernant le problème de la vérification.

tel-00004890, version 1 - 19 Feb 2004

$\models$	LTL	ALTL	PLTL	CLTL	PLTL$_\Box$ simple	PLTL$_\Box$	CLTL$_\Box$
Syst. de trans. finis	oui	oui	non	non	oui	oui	oui
Automates à pile	oui	oui	non	non	oui	oui	oui
processus PA	non	non	non	non	oui	non	non
réseaux de Petri	oui	oui	non	non	oui	oui	oui

Tab. 4.1 – *Décidabilité du problème de la vérification*

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Chapitre 5

Analyse d'automates communicants

Dans ce chapitre nous considérons le problème de la vérification pour les automates communicants (CFSM) introduit par Bochmann [Boc78]. Ce sont des automates finis qui communiquent entre eux via des canaux non-bornés gérés de manière FIFO. Ces systèmes ont le pouvoir expressif d'une machine de Turing. Nous ne pouvons donc pas obtenir des résultats de décidabilité pour leur problème de la vérification. Par contre, nous pouvons obtenir des méthodes de semi-décision. Nous appliquons la méthode de l'analyse symbolique aux automates communicants.

Les résultats dans ce chapitre ont été publiés dans [BH97].

5.1 Introduction

L'analyse du comportement d'un système se réduit souvent à un problème d'atteignabilité dans le modèle du système. Ce modèle est habituellement un système de transitions, qui peut être infini. Par exemple, si on veut vérifier que tous les comportements d'un système sont *sûrs*, il suffit de tester si l'ensemble des états *mauvais* n'est jamais atteint à partir des états initiaux. Cela peut se faire en calculant les états atteignables à partir des états initiaux et en vérifiant que l'intersection de cet ensemble avec l'ensemble des mauvais états est vide (voir Figure 5.1). Une autre manière de procéder consiste à tester si l'intersection de l'ensemble des bons états avec l'ensemble des états à partir desquels un mauvais état est atteignable est vide (voir Figure 5.2). Ces deux façons de reformuler le problème correspondent respectivement aux algorithmes d'analyse d'atteignabilité en avant et en arrière. Par conséquent, le calcul des prédécesseurs ou des successeurs d'un ensemble d'états S donné est un outil fondamental pour l'analyse des systèmes.

Soient S un ensemble d'états et $post(S)$ (resp. $pre(S)$) l'ensemble de successeurs (resp. prédécesseurs) immédiats de S. Soient $post^*(S)$ (resp. $pre^*(S)$) l'ensemble de tous les successeurs (resp. prédécesseurs) de S. Alors, $post^*(S)$ est la limite d'une séquence *infinie* croissante $(X_i)_{i\geq 0}$ donnée par $X_0 = S$ et $X_{i+1} = X_i \cup post(X_i)$ pour chaque $i \in I\!N$. De manière similaire, $pre^*(S)$ est la limite d'une séquence infinie en considérant la fonction pre au lieu de $post$. En partant de ces définitions nous pouvons dériver une procédure itérative

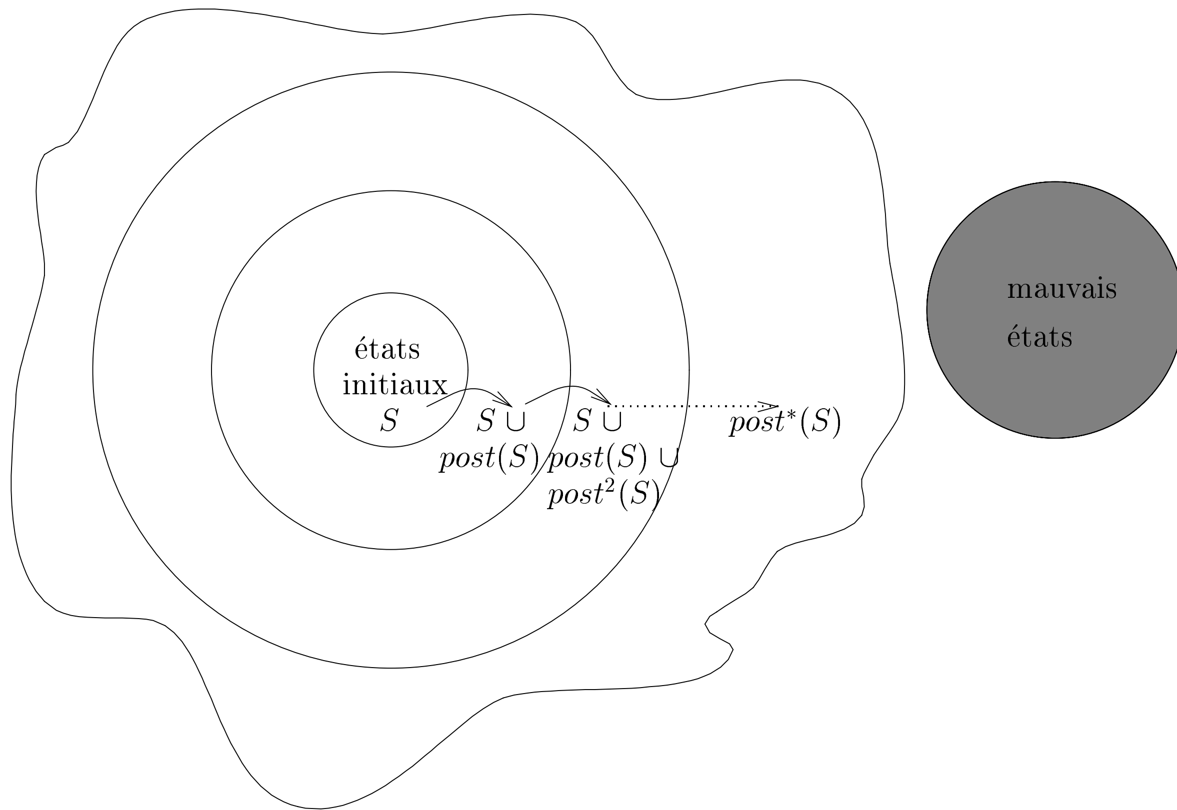


Fig. 5.1 – *Analyse en avant*

tel-00004890, version 1 - 19 Feb 2004

pour calculer l'ensemble $post^*(S)$ (resp. $pre^*(S)$). Cette procédure consiste simplement à calculer les éléments de la séquence $(X_i)_{i\geq 0}$ et à tester si $X_{i+1} = X_i$. Dans ce cas, la limite de la séquence est atteinte. Pour des systèmes qui sont modélisés par un automate d'état fini ces algorithmes terminent toujours, puisque les ensembles d'états X_i's sont finis. Néanmoins, plusieurs techniques symboliques ont été développées pour représenter de manière succincte des ensembles finis d'états de grande taille, par exemple les diagrammes de décision binaire (BDD) [Ake78, Bry86].

Pour des systèmes plus généraux qui sont modélisés par des automates avec des structures de données non-bornées (les variables entières, les variables réelles, les piles, les files, etc.), les ensembles X_i's sont en général infinis et il n'est pas sûr que la séquence $(X_i)_{i\geq 0}$ converge dans un nombre fini de pas. Pour vérifier ces systèmes nous devons trouver des structures *finies* qui permettent de représenter les ensembles *infinis* d'états auxquels nous nous intéressons. En sus, pour appliquer les algorithmes d'analyse en avant (resp. en arrière) ces structures devraient être effectivement fermées au moins par union, intersection et par les fonctions *post* et *pre*. Pour tester si la séquence converge, nous avons besoin de tester l'inclusion sur des ensembles d'états et nous voulons aussi tester si un ensemble est vide. Par conséquent, les problèmes d'inclusion et du vide pour ces structures doivent être

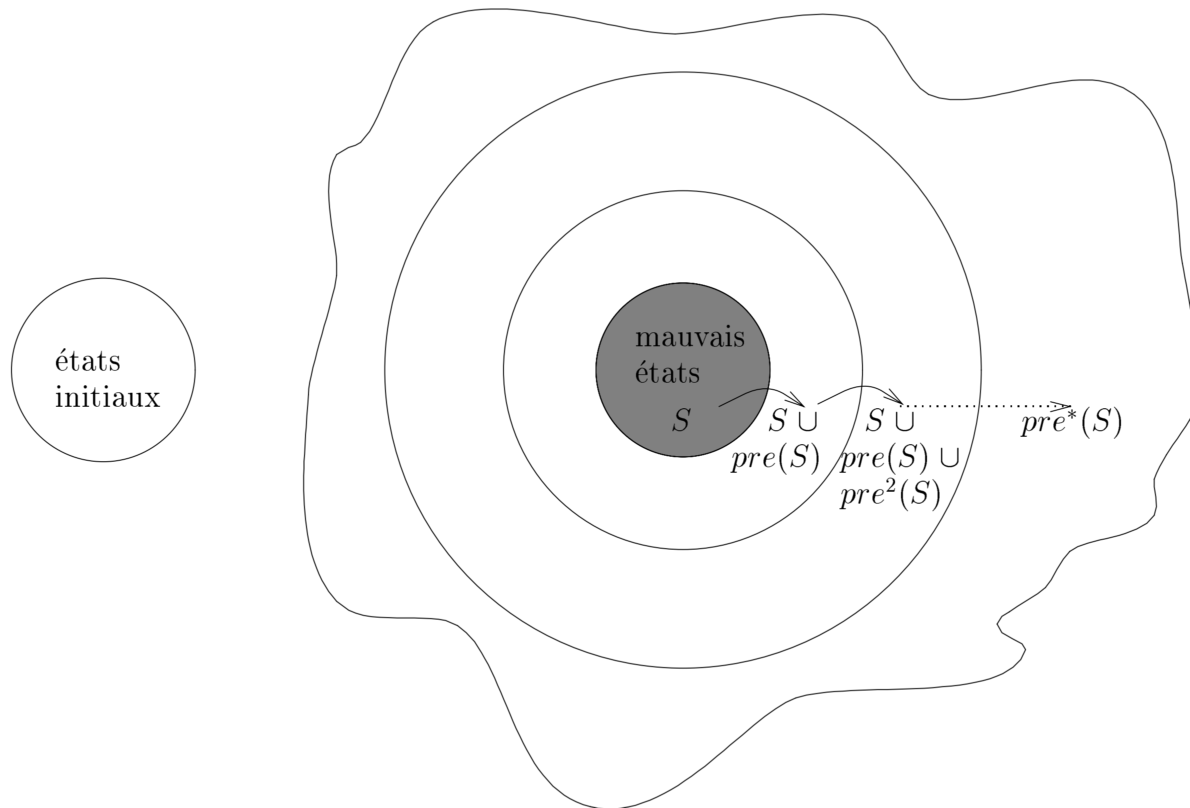


FIG. 5.2 – *Analyse en arrière*

tel-00004890, version 1 - 19 Feb 2004

décidables.

Par exemple, pour des systèmes qui manipulent des variables entières ou réelles comme les réseaux de Petri (systèmes à addition de vecteurs) ou les automates temporisés et hybrides, des structures de représentations basées sur les polyèdres convexes ou les ensembles de contraintes linéaires ont été considérées (voir [AD94, ACH+95, BW94, ME96]). Dans les systèmes qui manipulent des structures de données séquentielles comme des piles ou des files, les ensembles d'états (configurations) sont des vecteurs de mots. Pour ces systèmes on peut utiliser de manière naturelle des structures basées sur les automates finis (voir [BM96, BEM97, BG96, FWW97])

Outre les exigences sur les structures expliquées ci-dessus, nous devons considérer le problème de la convergence de la séquence $(X_i)_{i\geq 0}$. En général cette séquence n'atteint jamais sa limite. Pour faire face à ce problème nous pouvons considérer *l'accélération exacte* du calcul de la limite.

Cette technique consiste à définir une autre séquence $(Y_i)_{i\geq 0}$ telle que $Y_i \supseteq X_i$ et $Y_i \subseteq \bigcup_{j\in\mathbb{N}} X_j$ pour tout $i \in \mathbb{N}$. Ce qui veut dire que $(X_i)_{i\geq 0}$ et $(Y_i)_{i\geq 0}$ ont la même limite. Si $(X_i)_{i\geq 0}$, alors $(Y_i)_{i\geq 0}$ converge aussi. Par contre, $(Y_i)_{i\geq 0}$ peut converger là où $(X_i)_{i\geq 0}$ ne converge pas.

Cette approche à été utilisée dans [BM96, BEM97] pour définir des algorithmes de *model-checking* pour les automates à pile et différentes logiques temporelles linéaires et arborescentes. Ces algorithmes utilisent comme structures de représentation pour les contenus de piles les automates d'états fini (alternant). Dans ce cas il est garanti que la séquence $(Y_i)_{i\geq 0}$ termine.

Pour l'analyse des CFSM Boigelot et Godefroid [BG96] (voir aussi [BGWW97]) proposent des structures, appelées *Diagrammes de décision de contenu de file* (QDD). Un ensemble de contenus de plusieurs files est représenté par un automate d'états fini. Contrairement aux automates à pile, l'ensemble de configurations d'un CFSM est en général non-régulier et par conséquent ne peut pas être représenté par un QDD. Puisque le problème d'atteignabilité est indécidable le mieux que l'on puisse faire est d'utiliser des techniques d'accélération afin d'augmenter la chance de terminaison.

Une technique d'accélération est introduite dans [BW94] pour des systèmes avec variables entières. Cette technique est adaptée dans [BG96] pour l'analyse des CFSM's. Elle est basée sur la notion de *meta-transition*. Une meta-transition correspond à un circuit dans le graphe de contrôle du système. Ce circuit doit avoir une forme spéciale. Les successeurs ou prédécesseurs d'un ensemble d'états par une meta-transition sont donnés par un nombre quelconque d'exécutions du circuit. Étant donné la séquence $(X_i)_{i\geq 0}$ avec $X_0 = S$ et $X_{i+1} = X_i \cup post(X_i)$, la séquence *accélérée* $(Y_i)_{i\geq 0}$ est donnée par $Y_0 = X_0$ et $Y_{i+1} = Y_i \cup post(Y_{i+1}) \cup post^*_\theta(Y_i)$. L'ensemble $post^*_\theta(Y_i)$ correspond à l'ensemble de tous les successeurs de Y_i après autant de répétitions du circuit θ que possible.

Le problème est de déterminer pour quels circuits θ la classe des structures de représentation est fermée par $post^*_\theta$. Dans [BG96] les circuits considérés sont restreints pour garantir que l'image d'un ensemble régulier par $post^*_\theta$ est aussi un ensemble régulier. [BGWW97] donnent une caractérisation de tous les circuits qui satisfont cette propriété.

Dans ce mémoire nous proposons une généralisation de l'approche de [BG96] en considérant l'accélération exacte en utilisant *tout* circuit dans le graphe de transition du système. Le problème principal de cette nouvelle approche vient du fait que l'ensemble des états atteignables par un circuit peut être non-régulier. Par exemple, considérons un CSFM avec 2 files. Un circuit qui envoie un message a à une file et le message b a une autre, génère (en partant de files vides) un ensemble d'états atteignables de la forme $\{(a^k, b^k) \ : \ k \in \mathbb{N}\}$. Par conséquent, nous avons besoin de structures de représentation qui sont plus expressives que les automates finis.

Nous proposons dans la section 5.3 une structure, appelée diagrammes de décision de contenu de files contraints (CQDD) qui combine une sous-classe des automates finis avec des contraintes arithmétiques linéaires sur le nombre de fois que des transitions dans leurs calculs accepteurs sont prises. Nous montrons dans les sections 5.3 et 5.4 que les CQDD's satisfont toutes les propriétés d'une "bonne" structure de représentation, c'est-à-dire ils sont fermées par union, intersection, *post* et leur problèmes d'inclusion et du vide sont décidables. Ensuite, nous montrons dans la section 5.4 le résultat principal que les CQDD's sont fermés par $post^*_\theta$ pour chaque circuit θ. Nous montrons aussi que la classe des images inverses des ensembles CQDD représentables est fermée par pre^*_θ pour chaque circuit θ. Ces deux résultats nous permettent de définir des algorithmes génériques d'atteignabilité

tel-00004890, version 1 - 19 Feb 2004

en avant et en arrière dans la section 5.5. Ces algorithmes sont paramétrés par un ensemble de circuits. À la fin de ce chapitre nous comparons nos résultats avec d'autre travaux.

5.2 Les automates communicants

Dans cette section nous définissons les automates communicants et nous donnons quelques notations utilisées plus tard pour l'analyse de ces systèmes.

5.2.1 Définitions

Nous considérons ici une généralisation des automates communicants (*communicating finite-state machines* (CFSM) définis dans [Boc78]. Un automate communicant est un automate fini qui peut envoyer et recevoir des messages sur un ensemble fini de canaux FIFO (*first in first out*). Dans la définition habituelle des automates communicants, une transition ajoute un message à la fin d'un canal où enlève un message du début du canal. Nous généralisons cette définition en permettant que l'automate peut ajouter et enlever des messages de plusieurs canaux simultanément.

Définition 5.1 (Automate Communicant) :

Un *automate communicant* est un quadruple (S, K, Σ, T) où

- S est un ensemble fini d'*états de contrôle*,
- K est un ensemble fini de *canaux FIFO*, aussi appelé *files*,
- Σ est un ensemble fini de *messages*,
- T est un ensemble fini de *transitions*

Chaque transition a la forme (s_1, op, s_2), où s_1 et $s_2 \in S$, et op est un ensemble fini d' *opérations de canaux* de la forme $\kappa_i!w$ ou $\kappa_i?w$ avec $\kappa_i \in K$ et $w \in \Sigma^*$ tel que pour chaque canal κ_i il y a au plus une étiquette $\kappa_i!w$ ou $\kappa_i?w$ dans op.

Dans notre modèle, il y a une seule structure de contrôle donné par S. Dans la définition d'origine de Bochmann [Boc78], il y a plusieurs automates finis. En considérant la composition parallèle de ces automates nous obtenons une structure de contrôle donné par un seul automate.

Définition 5.2 (Configuration) :

Une *configuration* de $\mathcal{M}$ est un paire $\gamma = (s, \vec{w})$ où s est un état de contrôle de S, et $\vec{w} = (w_1, \ldots, w_{|K|})$ est un vecteur à $|K|$-dim de mots (c.-à-d. un $|K|$-uplet dans $(\Sigma^*)^{|K|}$), où chaque w_i est le contenu du canal κ_i. Nous écrivons $Conf$ pour l'ensemble de toutes les configurations de $\mathcal{M}$, c'est-à-dire $Conf = S \times (\Sigma^*)^{|K|}$.

tel-00004890, version 1 - 19 Feb 2004

Définition 5.3 (Relation de transition globale) :

La *relation de transition globale* entre configurations est définie comme suit: Soient $\gamma = (s, w_1, \ldots, w_{|K|})$ et $\gamma' = (s', w'_1, \ldots, w'_{|K|})$ deux configurations et soit op l'ensemble d'opérations de canaux. Alors, $\gamma \overset{op}{\rightarrow} \gamma'$ si et seulement s'il existe une transition $(s_1, op, s_2) \in T$ telle que, pour chaque $i \in \{1, \ldots, |K|\}$,

- si $\kappa_i?w \in op$ alors $ww'_i = w_i$,
- si $\kappa_i!w \in op$ alors $w'_i = w_i w$,
- sinon $w'_i = w_i$.

Définition 5.4 (transition exécutable, successeur et prédécesseur immédiat) :

Soit $\tau = (s, op, s') \in T$ une transition. τ est dite *exécutable* pour $\gamma = (s, \vec{w})$ s'il existe $\gamma' = (s', \vec{w}')$ telle que $\gamma \overset{op}{\rightarrow} \gamma'$. Dans ce cas, γ' est le *successeur immédiat* de γ par τ, et γ est le *prédécesseur* de γ' par τ.

Définition 5.5 (fonctions de prédécesseurs et successeurs) :

Les fonctions de prédécesseurs et successeurs sont définies comme suit: pre_τ et $post_\tau$ sont des fonctions $2^{Conf} \rightarrow 2^{Conf}$, telles que, pour chaque ensemble C de configurations $pre_\tau(C)$ (resp. $post_\tau(C)$) est l'ensemble de prédécesseurs (resp. successeurs) immédiats des configurations dans C par la transition τ.

Définition 5.6 (pre et $post$) :

pre et $post$ sont des fonctions $2^{Conf} \rightarrow 2^{Conf}$ définies par

$pre(C) = \bigcup_{\tau \in T} pre_\tau(C)$ et $post(C) = \bigcup_{\tau \in T} post_\tau(C)$

La notion d'*exécutabilité* peut être généralisée facilement aux séquences de transitions, en particulier aux *circuits* du graphe de transitions qui correspond à T.

Définition 5.7 (Circuit) :

Une séquence θ de transition de T est un *circuit* si elle est de la forme

$$(s_0, op_0, s_1)(s_1, op_1, s_2) \cdots (s_{n-1}, op_n, s_n)$$

avec $s_0 = s_n$.

Définition 5.8 :

La définition de pre_τ and $post_\tau$ est généralisée aux séquences de transitions par:

$pre_{\tau_1 \ldots \tau_n} = pre_{\tau_1} \circ \ldots \circ pre_{\tau_n}$ et $post_{\tau_1 \ldots \tau_n} = post_{\tau_n} \circ \ldots \circ post_{\tau_1}$

tel-00004890, version 1 - 19 Feb 2004

Étant donnée une séquence de transitions θ, les fonctions pre^*_θ et $post^*_\theta$ sont les fermetures réflexives et transitives de pre_θ et $post_\theta$, respectivement. Cela veut dire, qu'étant donné un ensemble de configurations C, $pre^*_\theta(C)$ (resp. $post^*_\theta(C)$) est l'ensemble de prédécesseurs (resp. successeurs) de configurations de C obtenus par un nombre quelconque d'itérations de la séquence θ. Le nombre d'itérations ne peut être supérieur à un que si θ est un circuit.

Nous définissons les fonctions pre^* et $post^*$ comme la fermeture réflexive et transitive de pre et $post$. La fonction pre^* (resp. $post^*$) donne l'ensemble de tous les prédécesseurs (resp. successeurs) d'un ensemble donné de configurations.

Définition 5.9 (Concaténation de n-uplets de mots) :

> Étant donnés deux n-uplets de mots $\vec{u} = (u_1, \ldots, u_n)$ et $\vec{v} = (v_1, \ldots, v_n)$, $\vec{u}.\vec{v}$ est le multi-mot $(u_1.v_1, \ldots, u_n.v_n)$.

Définition 5.10 ($out(\tau)$) :

> Soit $\tau = (s, op, s')$ une transition d'un CFSM (S, K, Σ, T). Alors $out(\tau)$ est le $|K|$-uplet de mots $(w_1, \ldots, w_{|K|}) \in (\Sigma^*)^{|K|}$ tel que, pour chaque $i \in \{1, \ldots, |K|\}$, $w_i = w$ si $\kappa_i!w \in op$, sinon $w_i = \epsilon$.

Définition 5.11 ($in(\tau)$) :

> Soit $\tau = (s, op, s')$ une transition d'un CFSM (S, K, Σ, T). Alors $in(\tau)$ est le $|K|$-uplet de mots $(w_1, \ldots, w_{|K|}) \in (\Sigma^*)^{|K|}$ tel que, pour chaque $i \in \{1, \ldots, |K|\}$, $w_i = w$ si $\kappa_i?w \in op$, sinon $w_i = \epsilon$.

Intuitivement, $out(\tau)$ est la sortie du système vers les canaux en exécutant τ, tandis que $in(\tau)$ est l'entrée du système des canaux. Ces définitions peuvent être généralisées aux séquences de transitions: Pour $\theta = \tau_1 \cdots \tau_n$, $out(\theta) = out(\tau_1)out(\tau_2) \cdots out(\tau_n)$ et $in(\theta) = in(\tau_1)in(\tau_2) \cdots in(\tau_n)$.

5.3 Diagrammes de décision de contenu de files contraints (CQDD)

Dans cette section nous introduisons les structures de représentation pour des ensembles de contenus de files (ensembles de configurations d'un automate communicant). Ces structures, appelés CQDD, combinent des automates d'états finis *séquentiels* (c'est-à-dire sans circuits imbriquées) avec des contraintes linéaires sur le nombre de fois les transitions dans ces automates sont prises en acceptant un mot. Cette combinaison permet de représenter des ensembles non-réguliers de contenus de files. sur-ensemble Nous introduisons d'abord les *automates séquentiels* qui sont à la base des CQDD. Nous montrons des résultats intermédiaires sur ces automates qui sont utilisés plus tard. Nous définissons ensuite les CQDD et nous montrons des propriétés de fermeture et la décidabilité des problèmes de l'inclusion et du vide.

tel-00004890, version 1 - 19 Feb 2004

5.3.1 Les automates séquentiels

5.3.1.1 Définitions

Dans cette section nous définissons les structures de bases nécessaires pour représenter des ensembles de configurations d'un automate communicant.

Définition 5.12 (Automate séquentiel) :

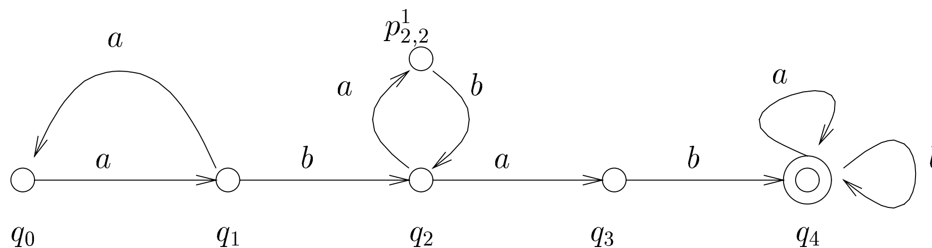
Un *automate séquentiel* (SA) sur Σ est un automate d'états fini $A = (Q, \Sigma, q_0, \rightarrow, \{q_m\})$ muni d'un ensemble fini d'indices $I \subseteq I\!N^2$ où

- $I \subseteq I\!N^2$ tel que
  - $\forall (i,j) \in I.\ i \leq j \leq m.$
  - $\forall (i,j) \in I.\ \forall (i_1, j_1) \in I.\ ((i_1, j_1) \neq (i,j) \Rightarrow (i_1 > j \vee j_1 < i))$
- Q est un ensemble fini d'états tel que $Q = \{q_0, q_1, \ldots, q_m\} \cup \bigcup_{(i,j)\in I} P_{i,j}$ et $P_{i,j} = \{p^1_{i,j}, \ldots, p^{\ell_i}_{i,j}\}$,
- $\rightarrow \subseteq Q \times \Sigma \times Q$ est l'ensemble de transitions défini comme étant le plus petit ensemble tel que (nous écrivons $q \xrightarrow{a} q'$ au lieu de $(q, a, q') \in \rightarrow$) :
  1. $\forall i \in \{0, \ldots, m \Leftrightarrow 1\}.\ \exists! a \in \Sigma.\ q_i \xrightarrow{a} q_{i+1}$,
  2. $\forall (i,j) \in I$, si $P_{i,j} \neq \emptyset$ alors
     - $\exists! a \in \Sigma.\ q_j \xrightarrow{a} p^1_{i,j}$,
     - $\forall k \in \{1, \ldots, \ell_i \Leftrightarrow 1\}.\ \exists! a \in \Sigma.\ p^k_{i,j} \xrightarrow{a} p^{k+1}_{i,j}$,
     - $\exists! a \in \Sigma.\ p^{\ell_i}_{i,j} \xrightarrow{a} q_i$
  3. $\forall (i,j) \in I$, si $P_{i,j} = \emptyset$ alors $\exists! a \in \Sigma$ avec $q_j \xrightarrow{a} q_i$
  4. si $P_{m,m} = \emptyset$ et $(m,m) \in I$ alors $\exists! \Sigma' \subseteq \Sigma.\ \forall a \in \Sigma'.\ q_m \xrightarrow{a} q_m$
- l'*état final* unique est q_m.

Les automates séquentiels sont des automates qui sont des séquences de circuits. Chaque état apparaît au plus dans un circuit, sauf peut-être le dernier état.

Exemple 5.1 :

Un automate séquentiel:



tel-00004890, version 1 - 19 Feb 2004

Cet automate reconnaît le langage $a(aa)^*b(ab)^*ab(a+b)^*$.

Définition 5.13 (Automate séquentiel restreint) :

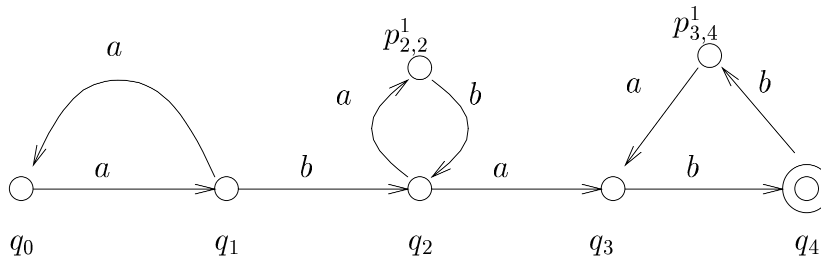
Un *automate séquentiel restreint* (RSA) est un automate séquentiel tel que le point 4 de la définition d'un automate séquentiel est remplacé par:

$$4'. \text{ si } P_{m,m} = \emptyset \text{ et } (m,m) \in I \text{ alors } \exists! a \in \Sigma.\ q_m \xrightarrow{a} q_m.$$

Un automate séquentiel restreint est un automate séquentiel où le dernier état a le même statut que les autres.

Exemple 5.2 :

Un automate séquentiel restreint:



Cet automate accepte le langage $a(aa)^*b(ab)^*ab(abb)^*$

Dans les automates séquentiels le degré de sortie des états q_i, sauf peut-être q_m, est au plus deux, tandis que le degré de sortie des états dans les P_i (resp. $P_{i,j}$) est toujours un.

Définition 5.14 (circuit) :

Étant donné un couple $(i,j) \in I$, le *circuit* défini par (i,j) est la séquence de transitions

$$(q_i, b_{i+1}, q_{i+1}) \cdots (q_{j-1}, b_j, q_j)(q_j, a_0, p^1_i)(p^1_{i,j}, a_1, p^2_{i,j}) \cdots (p^{\ell_i}_{i,j}, a_{\ell_i}, q_i)$$

Nous appelons q_i la racine de ce circuit. Le *circuit forme* le mot $b_{i+1} \ldots b_j a_0 \ldots a_{\ell_i}$.
Un *circuit simple* est un circuit défini par un couple dans I de la forme (i,i).

L'état q_m a un statut particulier parce qu'il peut être la racine de plusieurs circuits, mais dans ce cas tous les circuits doivent être des boucles (voir Définition 2.24). Dans les RSA l'état q_m a le même statut que les autres q_i's.

Des choix non-déterministes peuvent seulement apparaître dans les états q_i.

Définition 5.15 (Automate séquentiel déterministe) :

Nous écrivons DSA (resp. DRSA) pour un SA (resp. RSA) déterministe.

Les RSA's acceptent des langages sur Σ qui sont définissables par des expressions régulières de la forme $u_1 v_1^* u_2 v_2^* \cdots u_n v_n^* u_{n+1}$ où les u_i's et les v_i's sont des mots sur Σ tels que seuls u_1 et u_{n+1} peuvent être vides. Les SA's acceptent des mots $u_1 v_1^* u_2 v_2^* \cdots u_n (a_1 + \ldots + a_\ell)^*$ où les a_i's sont des symboles de Σ.

tel-00004890, version 1 - 19 Feb 2004

5.3.1.2 Résultats sur les automates séquentiels

Dans cette section nous présentons quelques résultats préliminaires sur les automates séquentiels.

Lemme 5.1 :

Soient A_1 un DRSA et A_2 un DSA. Alors il existe un DRSA A' tel que $L(A') = L(A_1) \cap L(A_2)$.

Preuve:
Soit $A' = A_1 \times A_2$ l'automate obtenu par la construction de produit d'automates classique. Il est évident que cet automate est un DRSA (en enlevant toutes les transitions et états inutiles).
□

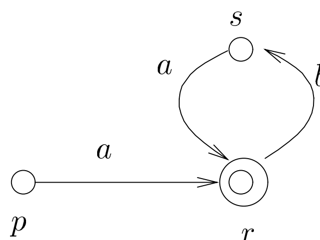
Lemme 5.2 :

Soit A un DRSA. Alors il existe un $k \in I\!N$ et des DSA's $A_1, \ldots, A_k$ tels que $\Sigma^* \setminus L(A) = \cup_{i=1}^{k} L(A_k)$.

Preuve:
Nous utilisons la construction classique pour la complémentation d'un automate. Pour obtenir une union de DSA nous le modifions en ajoutant un état pour chaque lettre et chaque état qui n'a pas de transitions sortantes avec cette lettre. Nous montrons d'abord la construction dans un exemple:

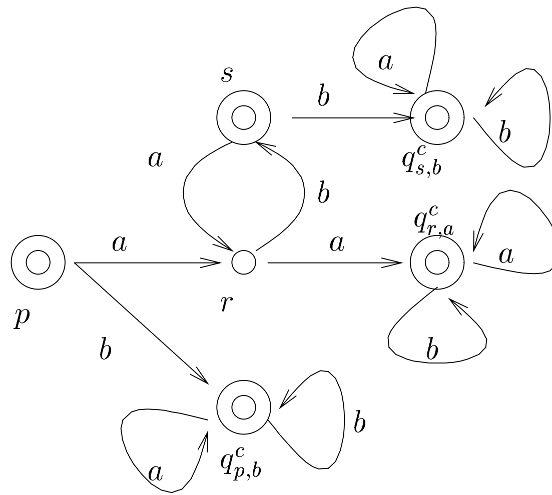
Exemple 5.3 :

Soit $\Sigma = \{a, b\}$ et A donné par:



tel-00004890, version 1 - 19 Feb 2004

Alors un automate qui reconnaît $\Sigma^* \setminus L(A)$ est donné par:



Il est évident que le langage de cet automate peut être décrit par une union de langages donnés par des DSA's.

Suite de la preuve: Soit $A = (Q, \Sigma, q_0, \rightarrow, q_m)$. Nous construisons d'abord un automate complet $A^c = (Q^c, \Sigma, q_0^c, \rightarrow^c, F)$ de la façon suivante:

- $Q^c = Q \cup \{q^c_{q,a} | q \in Q \wedge a \in \Sigma \wedge \not\exists q' \in Q. q \xrightarrow{a} q'\}$,
- $q_0^c = q_0$,
- $\rightarrow^c$ est le plus petit ensemble qui satisfait:
  - $\rightarrow^c \supseteq \rightarrow$
  - $\forall b \in \Sigma.\ q^c_{q,a} \xrightarrow{b}{}^c q^c_{q,a}$
  - $\forall q^c_{q,a} \in Q^c.\ q \xrightarrow{a}{}^c q^c_{q,a}$
- $F = Q^c \setminus \{q_m\}$

Finalement nous obtenons pour chaque état accepteur de A^c un automate A_i qui reconnaît le langage accepté par cet état accepteur. Cet automate n'est peut-être pas un DSA mais nous pouvons couper toutes les transitions qui mènent vers un état à partir duquel l'état final n'est pas atteignable. Nous obtenons un DSA. □

Lemme 5.3 :

Soient A_1 et A_2 deux DRSA. Alors il existe un ensemble fini de DRSA's tel que l'union de leurs langages est égal à $L(A_2)^{-1}.L(A_1)$ et il existe un ensemble fini de DRSA's tel que l'union de leurs langages est égal à $L(A_2) \cap Pref(L(A_1))$

tel-00004890, version 1 - 19 Feb 2004

Preuve:
Soit $A_1 = (Q_1, \Sigma, q^1_{init}, \rightarrow_1, \{q^1_{fin}\})$ et $A_2 = (Q_2, \Sigma, q^2_{init}, \rightarrow_2, \{q^2_{fin}\})$. Soit $A_1 \times A_2$ le produit de A_1 et A_2 donné par la construction classique. Soit

$$F = \{q \in Q_1 \ : \ (q, q^2_{fin}) \text{ est un état de } A_1 \times A_2\}$$

Pour chaque $q \in F$, soit $\overleftarrow{A^q_{1,2}}$ l'automate qui est une copie de $A_1 \times A_2$ avec (q^1_{init}, q^2_{init}) comme état initial et (q, q^2_{fin}) comme état final et soit $\overrightarrow{A^q_1}$ une copie de A_1 tel que l'état initial est q. Les $\overleftarrow{A^q_{1,2}}$ et $\overrightarrow{A^q_1}$ sont des DRSA's. Clairement,

$$\bigcup_{q \in F} L(\overleftarrow{A^q_{1,2}}) = L(A_2) \cap Pref(L(A_1))$$

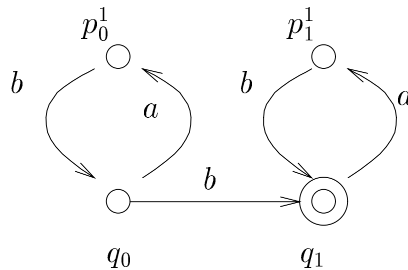
et

$$\bigcup_{q \in F} L(\overrightarrow{A^q_1}) = L(A_2)^{-1} \cdot L(A_1)$$

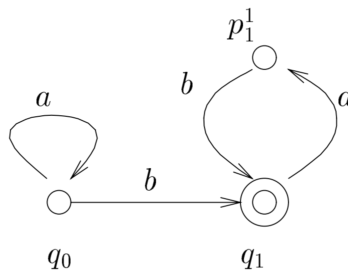
□

Exemple 5.4 :

Soit A_1 donné par:

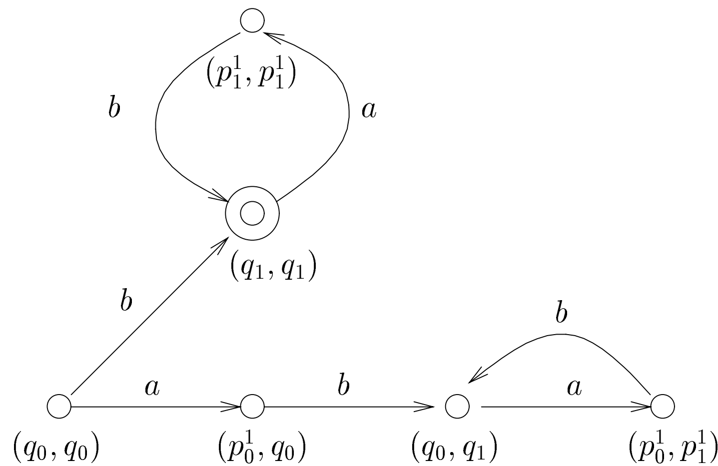


et A_2 donné par:



tel-00004890, version 1 - 19 Feb 2004

Alors $A_1 \times A_2$ est donné par:



et par exemple $\overrightarrow{A_1^{q_0}}$ est égal à A_1 et $\overleftarrow{A_{1,2}^{q_0}}$ est donné par $A_1 \times A_2$ où (q_0, q_1) est l'état final.

5.3.2 Formule caractéristique

Les structures de représentation pour les ensembles de configurations d'un automate communicant mélangent des structures d'automates séquentiels avec des contraintes linéaires sur les transitions. Nous avons besoin de caractériser les calculs accepteurs d'un automate séquentiel par une formule de Presburger (la définition d'une formule de Presburger se trouve dans section 2.1).

Soit $A = (Q, \Sigma, q_0, \rightarrow, q_m)$ un automate séquentiel. Soit X_A l'ensemble de variable $\{x_t \ : \ t \in \rightarrow\}$. Nous considérons pour chaque calcul ρ de A la valuation ν_ρ des variables de X_A telle que, pour chaque $x_t \in X$, $\nu_\rho(x) = |\rho'|_t$ (ρ' est définie dans la définition 2.21).

Nous définissons la formule de Presburger $[A]$ sur X_A qui caractérise l'ensemble des valuations qui correspondent à tous les calculs de A.

Pour cela nous introduisons quelques définitions. Nous écrivons $\mathcal{T}$ pour l'ensemble de transitions $\{t \in \rightarrow \ : \ \exists i \in \{0, \ldots, m \Leftrightarrow 1\}.\ \exists a \in \Sigma.\ t = (q_i, a, q_{i+1})\}$. Pour chaque état $q \in Q$, nous désignons q^+ (resp. q^-) pour l'ensemble des transitions de la forme (q', a, q) (resp. (q, a, q')) pour $q' \in Q$ et $a \in \Sigma$.

Définition 5.16 (Formule caractéristique) :

Soit A un automate séquentiel. La *formule caractéristique* de A, représenté par $[A]$ est donné par:

$$(\bigwedge_{q \in Q \setminus \{q_0, q_m\}} \sum_{t \in q^+} x_t = \sum_{t \in q^-} x_t) \wedge (1 + \sum_{t \in q_0^+} x_t = \sum_{t \in q_0^-} x_t) \wedge (\sum_{t \in q_m^+} x_t = 1 + \sum_{t \in q_m^-} x_t) \tag{5.1}$$

tel-00004890, version 1 - 19 Feb 2004

Puisque chaque calcul accepteur d'un automate simple doit visiter au moins une fois les états $\{q_0, \cdots, q_m\}$ le lemme suivant suit facilement.

Lemme 5.4 :

Soit A un automate séquentiel. Alors, pour chaque valuation ν des variables de X, ν satisfait $[A]$ si et seulement s'il existe un calcul accepteur ρ de A tel que $\nu = \nu_\rho$.

Il est bien connu que pour chaque automate fini nous pouvons construire une formule de Presburger caractéristique grâce au théorème de Parikh [Par66]. La formule que nous donnons ci-dessus est plus simple et exploite la structure des automates séquentiels.

5.3.3 Structures de représentation de contenus de files

Dans cette section nous introduisons les structures de représentation pour des ensembles de contenus de files. Ces structures combinent des automates d'états finis (automate séquentiel restreint déterministe) avec des contraintes linéaires sur le nombre de fois des transitions dans ces automates sont prises. Nous montrons ensuite des propriétés de fermeture et la décidabilité de l'inclusion et du problème du vide.

Définition 5.17 (automates séquentiels avec contraintes) :

Pour chaque $n \geq 1$, un *automate séquentiel avec contraintes* (CSA) est un ensemble de *composantes d'acceptation* $\mathcal{C} = \{\langle \mathcal{A}_1, f_1 \rangle, \ldots, \langle \mathcal{A}_m, f_m \rangle\}$ où, pour chaque $i \in \{1, \ldots, m\}$,

- $\mathcal{A}_i$ est un n-uplet de n automates séquentiels $(A_1^i, \ldots, A_n^i)$ sur Σ,
- f_i est une formule de Presburger sur l'ensemble de variables V_i qui contient l'ensemble $X_i = \{x_t \ : \ t \in \mathcal{T}_i\}$, où $\mathcal{T}_i$ est l'ensemble de toutes les transitions des automates dans $\mathcal{A}_i$, i.e., $\mathcal{T}_i = \bigcup_{j=1}^{n} \rightarrow_j^i$.

Le CSA $\mathcal{C}$ accepte un *multi-langage* à n dimensions, i.e. un ensemble de n-uplets de mots.

Définition 5.18 (multi-langage accepté par une composante) :

Pour chaque $i \in \{1, \ldots, m\}$, Le multi-langage d'une composante d'acceptation $\langle \mathcal{A}_i, f_i \rangle$, noté par $L(\langle \mathcal{A}_i, f_i \rangle)$ est l'ensemble de n-uplets de mots $(w_1, \ldots, w_n) \in (\Sigma^*)^n$ pour lesquelles il y a des calculs accepteurs $(\rho_1, \ldots, \rho_n)$ des automates $(A_i^1, \ldots, A_i^n)$, tels que $\exists (V_i \setminus X_i)$. f_i est satisfaite par la valuation $(\nu_{\rho_1}, \ldots, \nu_{\rho_n})$ (i.e., la valuation qui associe à chaque variable x_t l'entier $|\rho'_1 \ldots \rho'_n|_t$, où $t \in \mathcal{T}_i$).

Définition 5.19 (multi-langage accepté par un CSA) :

Le multi-langage d'un CSA $\mathcal{C}$, noté par $L(\mathcal{C})$, est l'union $\bigcup_{i=1}^{m} L(\langle \mathcal{A}_i, f_i \rangle)$.

tel-00004890, version 1 - 19 Feb 2004

Notation 5.1 :

Nous utilisons l'abréviation tt pour la formule

$$\bigwedge_{t \in \mathcal{T}_i} x_t \geq 0 \tag{5.2}$$

c.-à-d. la formule qui n'impose aucune contrainte.

Définition 5.20 (CDSA) :

Un CDSA à n dimensions est un CSA à n dimensions tel que tous ses automates sont déterministes (DSA's).

Définition 5.21 (CQDD) :

Un CQDD à n dimensions est un CDSA à n dimensions tel que tous ses automates sont restreints (DRSA's).

Définition 5.22 (n-CQDD) :

Pour chaque $n \geq 1$, nous écrivons n-CQDD (resp. n-CDSA,) pour la classe de tous les CQDD's à n dimensions (resp. CDSA's à n dimensions).

Définition 5.23 (CQDD définissable) :

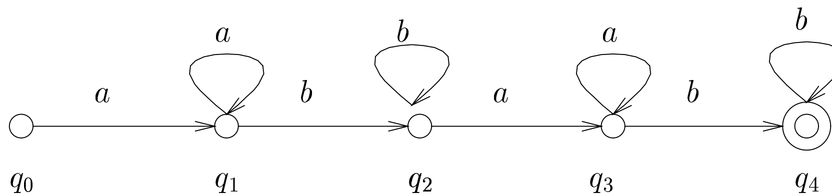
Un multi-langage à n dimensions $\mathcal{L}$ est CQDD (resp. CDSA) *définissable* s'il existe un CQDD à n dimensions (resp. CDSA) $\mathcal{C}$ tel que $L(\mathcal{C}) = \mathcal{L}$.

Définition 5.24 (CQDD inverse définissable) :

Un multi-langage à n dimensions $\mathcal{L}$ est CQDD (resp. CDSA) *inverse définissable* si son image inverse, noté $\mathcal{L}^R$, est CQDD (resp. CDSA) *définissable*.

5.3.4 Expressivité

Avec des CQDD's nous pouvons définir des langages non-réguliers. Par exemple, le langage dépendant du contexte $L_1 = \{a^n b^m a^n b^m \ : \ n, m \geq 1\}$. Pour définir L_1, nous utilisons l'automate A_1 donné par la figure suivante:

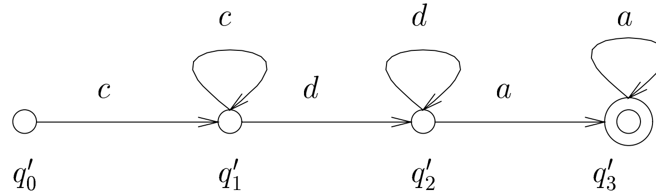


tel-00004890, version 1 - 19 Feb 2004

À chaque transition t de A_1 est associé une variable x_t. L_1 est définie par le CQDD à une dimension $\{\langle A_1, f_1\rangle\}$ où f_1 est:

$$x_{(q_1,a,q_1)} = x_{(q_3,a,q_3)} \quad \wedge \quad x_{(q_2,b,q_2)} = x_{(q_4,b,q_4)}$$

Considérons le multi-langage à deux dimensions $L_2 = \{(a^n b^m a^n b^m, c^m d^n a^m) \ : \ n, m \geq 1\}$. Pour définir ce multi-langage nous utilisons deux automates, l'automate A_1 défini précédemment et l'automate A_2 donné par la figure suivante:



L_2 est définie par le CQDD à deux dimensions $\{\langle(A_1, A_2), f_2\rangle\}$ où f_2 est:

$$x_{(q_1,a,q_1)} = n \ \wedge \ x_{(q_3,a,q_3)} = n \ \wedge \ x_{(q'_2,d,q'_2)} = n \ \wedge$$
$$x_{(q_2,b,q_2)} = m \ \wedge \ x_{(q_4,b,q_4)} = m \ \wedge \ x_{(q'_1,c,q'_1)} = m \ \wedge \ x_{(q'_3,a,q'_3)} = m$$

Ces exemples montrent que les CQDD's permettent d'exprimer des multi-langages non-réguliers qui comportent des contraintes sur le nombre d'occurrences de symboles à des positions qui peuvent être dans le même mot (comme dans L_1), où dans des mots différents (comme dans L_2). Cela permet de représenter des ensembles de contenus de files où il y a des contraintes (arithmétiques) qui relient les contenus de files différentes.

5.3.5 Opérations de base et problèmes de décision

Dans cette section, nous donnons et prouvons des résultats concernant les opérations booléennes sur les CQDD's.

Nous montrons d'une part que les CQDD's sont fermés par union et intersection et d'autre part que leur complémentation donne des CDSA's. En plus, l'intersection d'un CQDD avec un CDSA est un CQDD. En outre, nous montrons que les CQDD's sont fermés par concaténation et dérivation à gauche. Ces deux dernières opérations sont utilisées pour la construction des ensembles de successeurs et prédécesseurs pour un automate communicant (voir section 5.4). Finalement, nous montrons que les problèmes d'appartenance, du vide et d'inclusion sont décidables.

Pour simplifier la présentation nous donnons les preuves d'abord pour le cas des CQDD's à une dimension et une composante d'acceptation. À la fin de chaque preuve nous indiquons comment la généraliser aux CQDD's à n dimensions. Toutes les preuves peuvent être généralisées facilement aux CQDD's avec plusieurs composantes d'acceptations.

tel-00004890, version 1 - 19 Feb 2004

5.3.5.1 Fermeture par union et intersection

Proposition 5.1 :

Pour chaque $n \geq 1$, la classe n-CQDD est fermée par union et intersection.

Preuve:
La fermeture par union est triviale (par définition). Considérons d'abord la fermeture par intersection pour les CQDD à une dimension. Considérons les deux CQDD $\mathcal{C}_1 = \{\langle A_1, f_1(V_1)\rangle\}$ et $\mathcal{C}_2 = \{\langle A_2, f_2(V_2)\rangle\}$ où, pour $i \in \{1,2\}$, V_i est un de $X_i = \{x_t \ : \ t \in \rightarrow_i\}$.

Soit $A_1 \times A_2$ l'automate produit de A_1 et A_2. qui accepte le langage $L(A_1) \cap L(A_2)$. (son état final unique est le couple d'états finals de A_1 et A_2). Cet automate est d'après le Lemme 5.1 un DRSA. Chaque transition de $A_1 \times A_2$ est composé d'une transition t_1 dans A_1 et une transition t_2 dans A_2. Soit $\langle t_1, t_2\rangle$ cette transition. Alors, il est facile de voir que $L(\mathcal{C}_1) \cap L(\mathcal{C}_2)$ est accepté par le CQDD:

$$\mathcal{C}_1 \times \mathcal{C}_2 = \{\langle A_1 \times A_2, f_{1,2}(X_{1,2} \cup V_1 \cup V_2)\rangle\} \tag{5.3}$$

où $X_{1,2} = \{x_{\langle t_1,t_2\rangle} \ : \ t_1 \in\rightarrow_1 \text{ et } t_2 \in\rightarrow_2\}$ et la formule $f_{1,2}$ est donnée par:

$$f_1 \wedge f_2 \wedge \bigwedge_{t_1 \in\rightarrow_1} (x_{t_1} = \sum_{t_2\in\rightarrow_2} x_{\langle t_1,t_2\rangle}) \wedge \bigwedge_{t_2\in\rightarrow_2} (x_{t_2} = \sum_{t_1\in\rightarrow_1} x_{\langle t_1,t_2\rangle}) \tag{5.4}$$

Généralisation de la preuve à n dimensions:

Soit $\mathcal{C}_1 = \{\langle (A_1^1, \ldots, A_1^n), f_1(V_1)\rangle\}$ et $\mathcal{C}_2 = \{\langle (A_2^1, \ldots, A_2^n), f_2(V_2)\rangle\}$ où pour $i \in \{1,2\}$, V_i est un sur-ensemble de $X_i = \{x_t \ : \ t \in \cup_{j=1}^n \rightarrow_i^j\}$. Alors, $L(\mathcal{C}_1) \cap L(\mathcal{C}_2)$ est accepté par le CQDD:

$$\mathcal{C}_1 \times \mathcal{C}_2 = \{\langle (A_1^1 \times A_2^1, \ldots A_1^n \times A_2^n), f_{1,2}(X_{1,2} \cup V_1 \cup V_2)\} \tag{5.5}$$

où $X_{1,2} = \cup_{j=1}^n \{x_{\langle t_1^j, t_2^j\rangle} \ : \ t_1^j \in\rightarrow_1^j \text{ et } t_2^j \in\rightarrow_2^j\}$ et la formule $f_{1,2}$ est donnée par:

$$f_1 \wedge f_2 \wedge \bigwedge_{j=1}^n (\bigwedge_{t_1^j\in\rightarrow_1^j} (x_{t_1^j} = \sum_{t_2^j\in\rightarrow_2^j} x_{\langle t_1^j,t_2^j\rangle})) \wedge \bigwedge_{j=1}^n (\bigwedge_{t_2^j\in\rightarrow_2^j} (x_{t_2^j} = \sum_{t_1^j\in\rightarrow_1^j} x_{\langle t_1^j,t_2^j\rangle})) \tag{5.6}$$

□

D'après le Lemme 5.1 nous savons que le produit d'un DRSA avec un DSA est un DRSA. Par conséquent, la même construction nous permet de déduire:

Proposition 5.2 :

Pour chaque $n \geq 1$, l'intersection d'un n-CQDD avec un n-CDSA est un n-CQDD.

tel-00004890, version 1 - 19 Feb 2004

5.3.5.2 Complémentation

Proposition 5.3 :

Pour chaque $n \geq 1$, le complément d'un n-CQDD est un n-CDSA.

Preuve:
Considérons le CQDD $\mathcal{C} = \{\langle A, f(V)\rangle\}$ où V est un sur-ensemble de $X = \{x_t \ : \ t \in \rightarrow\}$. D'après le Lemme 5.2 le complément d'un DRSA est un DSA i.e., $\overline{L(A)}$ peut être écrit comme $\bigcup_{i=1}^{m} L(A_i)$ où chaque A_i est un DSA. Soit $\rightarrow_i$ la relation de transition de A_i et soit $X_i = \{x_t \ : \ t \in \rightarrow_i\}$. Alors, le langage $\overline{L(\{\mathcal{C}\})}$ est accepté par le CDSA:

$$\overline{\mathcal{C}} = \{\langle A_i, tt\rangle \ : \ 1 \leq i \leq m\} \cup \{\langle A, \neg\exists(V \setminus X).\ f\rangle\} \tag{5.7}$$

Il est important ici que l'automate A soit déterministe, i.e. chaque mot a exactement un calcul.

Généralisation de la preuve à n dimensions:

Dans ce cas, un multi-mot $(w_1, \ldots, w_n)$ n'est pas dans le langage d'un n-CQDD donné si

- au moins une composante w_i du multi-mot n'est pas dans le langage de l'automate pour cette composante.
- toutes les composantes w_i sont acceptées par les automates correspondants mais ne satisfont pas la contrainte f.

Formellement:
Soit $\mathcal{C} = \{\langle (A^1, \ldots, A^n), f(V)\rangle\}$ où V est un sur-ensemble de $X = \{x_t \ : \ t \in \cup_{j=1}^{n} \rightarrow^j\}$. $\overline{L(A^j)}$ peut être écrit comme $\bigcup_{i=1}^{m_j} L(A_i^j)$ où chaque A_i^j est un DSA. Soit $\rightarrow_i^j$ la relation de transition de A_i^j et soit $X_i = \{x_t \ : \ t \in \cup_{j=1}^{n} \rightarrow_i^j\}$. Alors, le langage $\overline{L(\{\mathcal{C}\})}$ est accepté par le CDSA:

$$\begin{aligned}
\overline{\mathcal{C}} \ = \ & \{\langle (B^1, B^2, \ldots, B^n), tt\rangle : \\
& \qquad (\forall k \in \{1, \ldots, n\}(B^k = A^k \vee \exists j \in \{1, \ldots, m_j\}.B^k = A_k^j)) \\
& \qquad \wedge \ \exists k \in \{1, \ldots, n\}(B^k \neq A^k)\} \\
& \cup \ \{\langle (A^1, \ldots, A^n), \neg\exists(V \setminus X).\ f\rangle\}
\end{aligned}$$

□

tel-00004890, version 1 - 19 Feb 2004

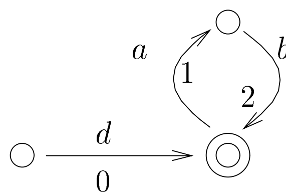
5.3.5.3 Fermeture par concaténation

Dans cette section nous montrons que les CQDD's sont fermées par concaténation.

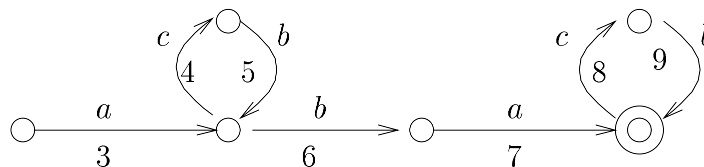
Définition 5.25 :

Soit $\mathcal{L}_1$ et $\mathcal{L}_2$ deux multi-langages à n dimensions. La *concaténation* de $\mathcal{L}_1$ et $\mathcal{L}_2$, notée $\mathcal{L}_1 \cdot \mathcal{L}_2$, est l'ensemble $\{\vec{w} \in (\Sigma^*)^n \ : \ \exists u \in \mathcal{L}_1. \exists v \in \mathcal{L}_2. \ \vec{w} = \vec{u}\vec{v}\}$.

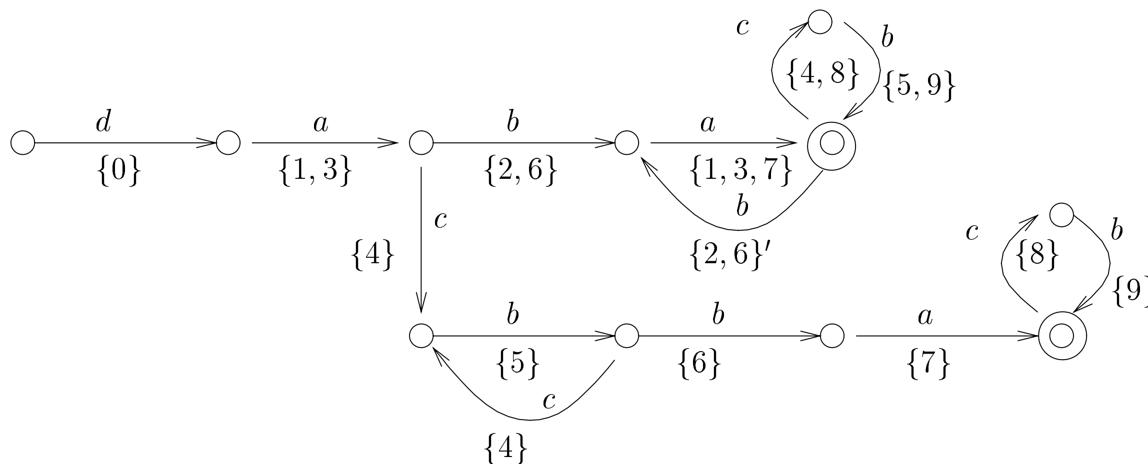
La concaténation de deux automates séquentiels restreints A_1 et A_2 peut donner un automate non-déterministe. Nous pouvons déterminiser cet automate et sommes confrontés au problème de définir une contrainte qui relie les variables associées aux transitions de cet automate avec les variables associées aux transitions de A_1 et A_2. Nous illustrons ce problème avec un exemple. Soient $\mathcal{C}_1 = \{\langle A_1, f_1(V_1)\rangle\}$ et $\mathcal{C}_2 = \{\langle A_2, f_2(V_2)\rangle\}$ deux CQDD's, où pour $i \in \{1,2\}$, $A_i = (Q_i, \Sigma, q^i_{init}, \rightarrow_i, q^i_{fin})$, et V_i est un sur-ensemble de $X_i = \{x_t \ : \ t \in \rightarrow_i\}$. Soit A_1 l'automate:



et A_2 l'automate



Si nous concaténons les deux automates (en identifiant l'état final de A_1 avec l'état initial de A_2) et si nous déterminisons avec la construction classique (*subset construction*) nous obtenons un automate A_d où chaque transition est un sous-ensemble de transitions de A_1 et A_2. Notez que dans cet exemple, les sous-ensembles $\{2, 6\}$ et $\{4\}$ apparaissent deux fois.



Soit $X_d = \{x_t \ : \ t \in \to_d\}$. Pour définir $L(\mathcal{C}_1) \cdot L(\mathcal{C}_2)$ nous devons associer à A_d une formule $f_d(V_d)$ (où V_d est un sur-ensemble de X_d). Étant donné un mot $w \in L(A_d)$ cette formule doit exprimer le fait qu'il existe $w_1 \in L(A_1)$ et $w_2 \in L(A_2)$ tels que $w = w_1.w_2$ et la valuation induite par le calcul de w_1 (resp. w_2) sur A_1 (resp. A_2) satisfait f_1 (resp. f_2). La formule f_d doit par conséquent relier les variables X_d avec les variables X_1 et X_2.

Dans l'exemple ci-dessus considérons deux mots acceptés par A_d:

- Le mot *dababacbbacb* $\in L(A_d)$ induit une valuation des variables: $x_{\{0\}} = x_{\{1,3\}} = x_{\{2,6\}} = 1$, $x_{\{2,6\}'} = x_{\{4,8\}} = x_{\{5,9\}} = 2$, $x_{\{1,3,7\}} = 3$ et toutes les autres variables valent 0. Cela correspond à *dabab* $\in L(A_1)$ et *acbbacb* $\in L(A_2)$ avec la valuation $x_0 = x_3 = x_4 = x_5 = x_6 = x_7 = x_8 = x_9 = 1$ et $x_1 = x_2 = 2$.
- Le mot *dabababacbcb* $\in L(A_d)$ induit la même valuation des variables X_d que le premier mot. *dabababacbcb* correspond à *dabab* $\in L(A_d)$ et *abacbcb* $\in L(A_2)$ avec la valuation $x_0 = x_3 = x_6 = x_7 = 1$, $x_1 = x_2 = x_8 = x_9 = 2$ et $x_4 = x_5 = 0$.

Cet exemple montre qu'une valuation des variables X_d peut correspondre à plusieurs valuations des variables X_1 et X_2.

Nous avons besoin d'une manière systématique de définir cette correspondance. Cela est difficile en général parce que

- un mot $w \in A_d$ peut parfois être divisé de manières différentes en deux mots w_1 et w_2 tels que $w = w_1.w_2$, $w_1 \in L(A_1)$ et $w_2 \in L(A_2)$,
- les transitions de A_d peuvent être étiquetées par le même sous-ensemble de transitions de A_1 et A_2,
- les transitions de A_1 ou A_2 peuvent apparaître plusieurs fois dans des transitions de A_d.

Dans ce qui suit nous donnons des lemmes techniques qui montrent que nous pouvons toujours transformer les automates A_1 et A_2 de sorte que l'automate A_d calculé ait une structure qui nous permette de définir facilement la correspondance entre X_1,X_2 et X_d.

Lemme 5.5 :

> Soit $\mathcal{C} = \{\langle A, f(V)\rangle\}$ un CQDD tel que l'état final de A est dans un circuit. Alors, il existe un CQDD $\mathcal{C}' = \{\langle A', f'(V')\rangle\}$ avec $L(\mathcal{C}) = L(\mathcal{C}')$ et l'état final de A' est dans un circuit simple.

Preuve:
Soit I l'ensemble d'indices associé à $A = (Q, \Sigma, q_0, \to, \{q_m\})$. Si le dernier circuit de A n'est pas simple, alors il existe une paire $(i, m) \in I$ tel que les états $q_i, \ldots, q_m \in Q$ sont dans le dernier circuit. Un calcul accepteur visite au moins une fois ces états. Soit A'' l'automate construit à partir de A en déroulant le dernier circuit jusqu'à l'état q_m, c'est-à-dire

tel-00004890, version 1 - 19 Feb 2004

les états $q_i, \ldots, q_{m-1}$ sont copiés et la relation de transition est modifiée de telle manière que la première visite de ces états se fait en dehors du dernier circuit et l'autre dedans. Maintenant, l'état final de A'' est dans un circuit *simple*. Clairement $L(A) = L(A'')$. Soit $\mathcal{C}' = \{\langle A', f'(V')\rangle\} = \mathcal{C} \times \{\langle A'', tt\rangle\}$. Puisque nous n'imposons pas de contraintes sur A'', nous avons $L(\mathcal{C}) = L(\mathcal{C}')$. L'état final de A' est dans un circuit simple et l'opération $\times$ construit f' à partir de f. □

Nous montrons que chaque CQDD peut être décomposé dans une union fini de CQDD's dont les automates ont un dernier circuit d'une taille arbitrairement grande.

Lemme 5.6 :

> Soit $\mathcal{C} = \{\langle A, f(V)\rangle\}$ un CQDD tel que l'état final de A est dans un circuit. Soit w le mot formé par ce circuit. Alors, $\forall n \in I\!N$. $\forall i \in I\!N$ avec $(0 \leq i < n)$ il existe des CQDD's $\mathcal{C}_i = \{\langle A_i, f_i(V_i)\rangle\}$ tels que le dernier circuit de chaque A_i est simple, contient l'état final, et forme le mot w^n. En plus, $L(\mathcal{C}) = \bigcup_{i=0}^{n-1} L(\mathcal{C}_i)$.

Preuve:
Soit $\mathcal{C} = \{\langle A, f(V)\rangle\}$ un CQDD avec $A = (Q, \Sigma, q_0, \rightarrow, \{q_m\})$ et un ensemble d'indices I tel que l'état final est dans un circuit qui forme le mot w. D'après le Lemme 5.5 nous pouvons supposer que ce circuit est simple. Soit $n \in I\!N$. À partir de A nous construisons n automates A'_i tels que $L(A) = \bigcup_{i=0}^{n-1} L(A'_i)$ et tels que le dernier circuit des A'_i's forme le mot w^n: Nous obtenons l'automate A'_i à partir de A en déroulant le dernier circuit i fois et le dernier circuit de A'_i forme le mot w^n au lieu de w. Le reste de l'automate n'est pas modifié. Pour chaque i avec $0 \leq i < n$ les CQDD's $\mathcal{C}_i$ sont donnés par $\mathcal{C} \times \{\langle A'_i, tt\rangle\}$. □

Lemme 5.7 :

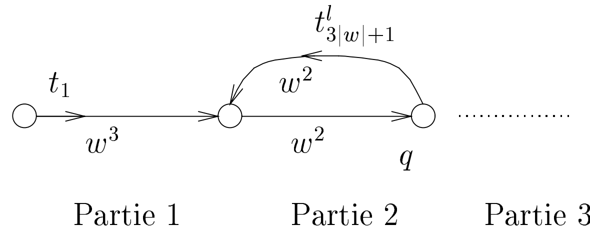
> Soit $\mathcal{C} = \{\langle A, f(V)\rangle\}$ un CQDD tel qu'il existe $k_1, k_2 \in I\!N$ avec $k_2 > k_1$ et $w \in \Sigma^+$ tels que
>
> – les premiers $k_1|w|$ états de A ne sont pas dans un circuit et forment le mot w^{k_1},
>
> – le premier circuit de A a $k_2|w|$ états et forme le mot w^{k_2}.
>
> Alors, il existe un CQDD $\mathcal{C}' = \{\langle A', f'(V')\rangle\}$ tel que $L(\mathcal{C}) = L(\mathcal{C}')$ et l'état initial de A' est dans un circuit de taille $k_2|w|$ qui forme le mot w^{k_2}.

Preuve:
Soit $A = (Q, \Sigma, q_0, \rightarrow, \{q_m\})$ et soit I l'ensemble d'indices associé. Intuitivement, nous devons faire correspondre les premiers $k_1|w|$ états de A avec les premiers $k_1|w|$ états de

tel-00004890, version 1 - 19 Feb 2004

son premier circuit. Soit $c_1 = k_1|w|$ et $c_2 = k_2|w|$. L'automate A consiste en trois parties: Premièrement, c_1 états qui ne sont pas dans un circuit. Deuxièmement, un circuit avec c_2 états et troisièmement, le reste de l'automate (qui peut être vide). Deux cas se présentent: Ou bien il y a au maximum une transition t à partir d'un état q de la deuxième partie vers la troisième ou alors la deuxième partie contient l'état final q. Dans ces deux cas, soit w' le mot formé par les transitions de A de l'état initial vers q.

Par exemple,

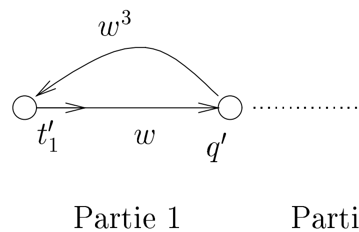


Ici, nous avons $w' = w^5$. Soient $t_1, \ldots, t_{c_1}$ les premières c_1 transitions de A et $t^l_1, \ldots, t^l_{c_2}$ les c_2 transitions du premier circuit de A et $\mathcal{T}$ l'ensemble de toutes les autres transitions. $\mathcal{T}$ peut être vide.

Nous construisons un automate A' de la manière suivante: La première partie contient l'état initial dans un circuit de taille c_2 qui forme le mot w^{k_2}. Soit q' l'état obtenu en déroulant w' sur ce circuit à partir de l'état initial. Puisque $|w'| < c_1 + c_2 < 2c_2$ (du fait que $k_2 > k_1$) le calcul de w' sur ce circuit visite l'état initial au maximum deux fois.

- Si l'état q de A défini ci-dessus est l'état final de A, alors l'état final de A' est q'.
- S'il y a une transition t de l'état q dans A vers la troisième partie de A, alors à partir de q' il y a la même transition dans A' et la deuxième partie de A' est une copie de la troisième partie de A.

  Par exemple,



Il est clair que A' est un DRSA. Pour finir la construction du CQDD nous devons définir la formule $f'(V')$: Une exécution d'une transition dans le premier circuit de A' correspond soit à une exécution d'une transition dans la première partie de A soit à l'exécution d'une transition de la deuxième partie (le premier circuit) de A. En plus, nous devons exprimer le fait que les nombres d'exécutions des transitions de A doivent correspondre à un calcul accepteur dans A et qu'ils satisfont f. Cela peut être exprimé par la contrainte $f'(V')$. Soient

tel-00004890, version 1 - 19 Feb 2004

$t'_1, \ldots, t'_{c_2}$ les transitions du premier circuit de A' et $\mathcal{T}'$ l'ensemble des autres transitions. Alors, $f'(V')$ est donnée par:

$$f(V) \wedge [A] \wedge \bigwedge_{i=1}^{c_1} (x_{t'_i} = x_{t_i} + x_{t^l_{(i+c_1) mod\ c_2}}) \wedge \bigwedge_{i=c_1+1}^{c_2} (x_{t'_i} = x_{t^l_{(i+c_1) mod\ c_2}}) \wedge \bigwedge_{t \in \mathcal{T}} x_t = x_{t'}$$

Dans l'exemple, nous avons entre autre $x_{t'_1} = x_{t_1} + x_{t^l_{3|w|+1}}$.

La contrainte f' peut imposer, que le premier circuit de A' doit être pris au moins une fois. Cela correspond au cas, où le déroulement de w' sur le circuit visite deux fois l'état initial (voir exemple ci-dessus).

Il est facile de voir que $L(\mathcal{C}) = L(\mathcal{C}')$. □

Le Lemme suivant montre qu'un CQDD dont l'automate A satisfait une certaine condition peut être décomposé en deux CQDD's.

Lemme 5.8 :

Soit $\mathcal{C} = \{\langle A, f(V)\rangle\}$ un CQDD avec $A = (Q, q_0, \rightarrow, \{q_m\})$. Soient $k_1, k_2 \in I\!N$ avec $k_2 > k_1$ et $w \in \Sigma^+$ tels qu'il existe $q_1 \in Q$ avec $q_0 \overset{w^{k_1}}{\rightarrow} q_1 \overset{w^{k_2}}{\rightarrow} q_1$. Alors, il existe deux CQDD $\mathcal{C}'$ et $\mathcal{C}''$ avec $L(\mathcal{C}) = L(\mathcal{C}') \cup L(\mathcal{C}'')$ tels que

- pour tout automate A' dans $\mathcal{C}'$ nous avons
  - l'état initial est dans un circuit,
  - le circuit contenant l'état initial a la taille $k_2|w|$ et forme le mot w^{k_2}.
- pour tout automate A'' dans $\mathcal{C}''$, nous avons $w^{k_1} \notin Pref(L(A''))$.

Preuve:
Σ^* peut s'écrire $L \cup \overline{L}$ avec $L = w^{k_1}.(w^{k_2})^*.\Sigma^*$

Il est facile de voir que L peut être exprimé comme l'union de langages d'automates séquentiels déterministes $A'_1, \ldots, A'_{n'}$ tel que les premiers $k_1|w|$ états de ces automates ne sont pas dans un circuit et forment le mot w^{k_1}. En plus, le premier circuit a la taille $k_2|w|$ et forme le mot w^{k_2}. Alors, le produit des automates A et A'_i est un automate tel que les premiers $k_1|w|$ états de ces automates ne sont pas dans un circuit et forment le mot w^{k_1}. En plus, le premier circuit a la taille $k_2|w|$ et forme le mot w^{k_2}.

Alors, d'après le Lemme 5.7 nous pouvons transformer le CQDD $\mathcal{C}' = \mathcal{C} \times \{\langle A'_i, tt\rangle \ : \ 1 \leq i \leq n'\}$ tel que les automates ont la forme voulue.

$\overline{L}$ peut être écrit comme une union de langages d'automates séquentiels déterministes $A''_1, \ldots, A''_{n''}$. Alors, $\mathcal{C}''$ est donné par $\mathcal{C} \times \{\langle A''_i, tt\rangle \ : \ 1 \leq i \leq n''\}$. □

tel-00004890, version 1 - 19 Feb 2004

Lemme 5.9 :

Soit $\mathcal{C} = \{\langle A, f(V)\rangle\}$ un CQDD et $k \in I\!N$. Alors, il existe un CQDD $\mathcal{C}'$ tel que $L(\mathcal{C}) = L(\mathcal{C}')$ et pour tous les automates dans $\mathcal{C}'$ les premiers k états (s'ils existent) ne sont pas dans un circuit.

Preuve:
Il est facile de voir que Σ^* peut être écrit comme une union finie de langages d'automates séquentiels déterministes $A_1, \ldots, A_n$ tels que pour tous les automates A_i les premiers k états (s'ils existent) ne sont pas dans un circuit.

Alors, $\mathcal{C}'$ est donné par $\bigcup_{i=1}^{n}(\mathcal{C} \times \{\langle A_i, tt\rangle\})$. □

Avec tous ces lemmes techniques, nous sommes en mesure de prouver la proposition suivante.

Proposition 5.4 :

Pour tout $n \geq 1$, la classe n-CQDD est fermée par concaténation.

Preuve:
Considérons les deux CQDD's $\mathcal{C}_1 = \{\langle A_1, f_1(V_1)\rangle\}$ et $\mathcal{C}_2 = \{\langle A_2, f_2(V_2)\rangle\}$ où, pour $i \in \{1, 2\}$, $A_i = (Q_i, \Sigma, q^i_{init}, \rightarrow_i, q^i_{fin})$, et V_i est un sur-ensemble de $X_i = \{x_t \ : \ t \in \rightarrow_i\}$. Nous construisons un CQDD qui accepte $L(\mathcal{C}_1) \cdot L(\mathcal{C}_2)$.

Si l'état final de A_1 n'est pas dans un circuit nous concaténons les deux automates A_1 et A_2 (cela donne un automate séquentiel restreint déterministe) et nous prenons la conjonction des deux formules f_1 et f_2. Si l'état final est dans un circuit, l'automate que nous obtenons en concaténant A_1 et A_2 peut être non-déterministe. La déterminisation modifie sa structure et nous devons redéfinir les contraintes. Dans ce qui suit nous utilisons les lemmes précédents qui nous permettent de supposer que $\mathcal{C}_1$ et $\mathcal{C}_2$ ont une structure spéciale.

Soit w le mot donné par le circuit qui contient l'état final de A_1.

Nous considérons deux cas:

– Cas 1: $\exists k \in I\!N.\ \forall k' \geq k.\ w^{k'} \notin Pref(L(A_2))$
  Cette condition peut facilement être testée sur l'automate A_2 (voir Cas 2). Nous pouvons supposer par le Lemme 5.6 que

  (1) le dernier circuit de A_1 est simple, forme le mot w^k et a la taille $k|w|$.

  En plus, nous supposons par le Lemme 5.9 que

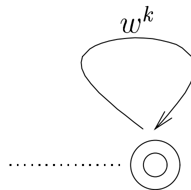
  (2) ou bien A_2 n'a pas de circuits et moins que $k|w|$ états ou alors les premiers $k|w|$ états ne sont pas dans un circuit.

tel-00004890, version 1 - 19 Feb 2004

La concaténation des deux DRSA A_1 et A_2 est un RSA (nous identifions l'état q^1_{fin} avec l'état q^2_{init}). Soit $A_1 \cdot A_2$ l'automate ainsi obtenu. Puisque les CQDD's sont définis en utilisant les automates séquentiels restreints déterministes nous devons déterminiser $A_1 \cdot A_2$. Nous utilisons pour cela la construction standard de sous-ensembles. Nous appelons l'automate obtenu par cette construction A_d. Chaque état de A_d est un sous-ensemble d'états de A_1 et A_2. À chaque transition de A_d est associé un sous-ensemble de $\rightarrow_1 \cup \rightarrow_2$. Examinons la structure de A_d en détail. Grâce aux conditions (1) et (2) A_d consiste en trois parties:

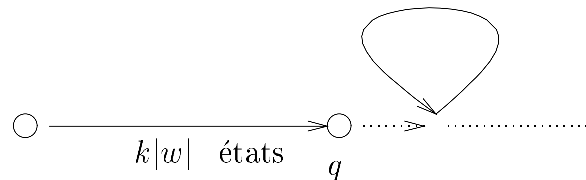
- Partie 1: Une copie de A_1 sans son dernier circuit. À chaque transition est associé une transition de A_1.
- Partie 2: Chaque état de cette partie est soit un ensemble qui consiste en un état de A_1 ou un état de A_1 et un des premiers $k|w|$ états de A_2. Cette partie forme un circuit avec racine $\{q^2_{init}\}$. À cause de $w^k \notin Pref(L(A_2))$ il y a au moins une transition de ce circuit à laquelle est associée exactement une transition de A_1, parce qu'à un certain point, A_2 est en désaccord avec le dernier circuit de A_1.
- Partie 3: Une copie du reste de A_2 (peut être vide, si A_2 n'a pas de circuits). À toutes les transitions sont associées une transition de A_2.

Il y a exactement une transition de la partie 1 vers la partie 2, de même que de la partie 2 vers la partie 3 (si elle existe). L'automate A_d est donc un DRSA. Nous illustrons cela par un exemple schématique: Soit A_1 donné par:

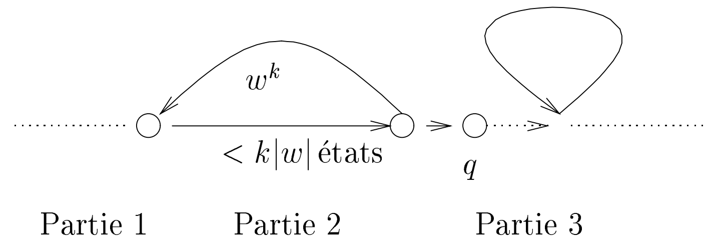


Reste de A_1

et A_2 donné par:



Alors A_d a la forme:



Le problème qui reste à traiter est de définir une formule pour cet automate. Soit $\rightarrow_d$ l'ensemble de transitions de A_d et $X_d = \{x_t \; : \; t \in \rightarrow_{A_d}\}$. Pour une transition $t \in \rightarrow_d$ soit $\lambda(t)$ l'ensemble de transitions associées de A_1 et A_2. Grâce aux conditions (1) et (2) chaque transition de A_1 et A_2 apparaît exactement une fois dans une transition de A_d. Par conséquent, l'exécution d'une transition de t dans A_d corresponds à l'exécution d'une des transitions dans $\lambda(t)$. Cela impose une contrainte exprimée par la formule $f_d(V_1 \cup V_2 \cup X_d)$:

$$\bigwedge_{t \in \rightarrow_d} (x_t = \sum_{t' \in \lambda(t)} x_{t'})$$

En sus, nous devons exprimer le fait que les nombres d'exécutions des transitions dans A_1 et A_2 correspondent à un calcul accepteur dans A_1 et A_2 et qu'il doivent satisfaire $f_1 \wedge f_2$. Alors, les contraintes sont données par la formule $f_{det}(V_1 \cup V_2 \cup X_d)$:

$$[A_1] \wedge [A_2] \wedge f_1 \wedge f_2 \wedge f_d(V_1 \cup V_2 \cup X_d)$$

et il est facile de voir que le langage $L(\mathcal{C}_1) \cdot L(\mathcal{C}_2)$ est accepté par le CQDD:

$$\mathcal{C}_1 \cdot \mathcal{C}_2 = \{\langle A_d, f_{det} \rangle\}$$

– Cas 2: $\forall n \in I\!N : w^n \in Pref(L(A_2))$
  Cette condition peut être facilement testée sur A_2. Puisque A_2 est un automate séquentiel déterministe, il existe nécessairement $k_1, k_2 \in I\!N$ avec $k_2 > k_1$ et $q_1 \in Q_2$ tels que $q^2_{init} \stackrel{w^{k_1}}{\rightarrow} q_1 \stackrel{w^{k_2}}{\rightarrow} q_1$ (pour garantir $k_2 > k_1$, le chemin $q_1 \stackrel{w^{k_2}}{\rightarrow} q_1$ peut correspondre à des répétitions du circuit).

  Par le Lemme 5.8 nous pouvons partager $\mathcal{C}_2$ en deux CQDD's:

  – $\mathcal{C}''_2$, où pour tout automate A'' nous avons $w^{k_1} \notin Pref(L(A''))$. Pour ces CQDD's nous pouvons raisonner comme dans le cas 1.
  – $\mathcal{C}'_2$, où tous les automates ont une forme spéciale.

  nous pouvons supposer que

  (1) l'état initial de A_2 est dans un circuit qui forme le mot w^{k_2} et a la taille $k_2|w|$.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

En sus, nous pouvons supposer (Lemme 5.6) que

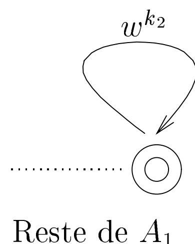
(2) le dernier circuit de A_1 est simple, forme le mot w^{k_2} et a la taille $k_2|w|$.

Comme dans le cas 1 nous construisons l'automate A_d et nous définissons $\rightarrow_d$, X_d et $\lambda(t)$. A_d consiste en trois parties:

- Partie 1: Une copie de A_1 sans le dernier circuit. À chaque transition est associée une transition de A_1
- Partie 2: La superposition du dernier circuit de A_1 et du premier circuit de A_2 (ce deux circuits ont exactement la même forme). Chaque état est un couple avec un état du dernier circuit de A_1 et un état du premier circuit de A_2.
- Partie 3: consiste en le reste de A_2 (peut être vide si l'état final de A_2 est dans le premier circuit.

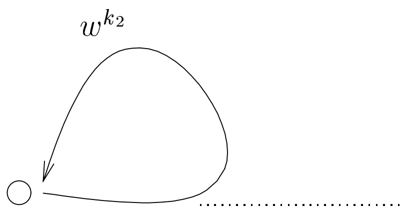
Il y a exactement une transition de la partie 1 vers la partie 2, de même que de la partie 2 vers la partie 3 (si elle n'est pas vide). A_d est donc un DRSA.

Nous illustrons cela avec un exemple schématique: Soit A_1 l'automate:

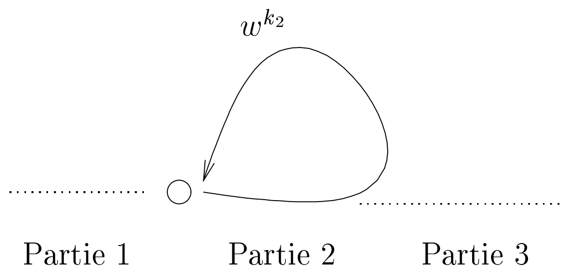


Reste de A_1

et A_2:



Alors, A_d a la forme:



Partie 1 Partie 2 Partie 3

Chaque transition de A_1 et A_2 apparaît exactement une fois dans une transition de A_d. Comme dans le cas 1 nous définissons $f_d(V_1 \cup V_2 \cup X_d)$:

$$\bigwedge_{t \in \to_d} (x_t = \sum_{t' \in \lambda(t)} x_{t'})$$

et $f_{det}(V_1 \cup V_2 \cup X_d)$:

$$[A_1] \wedge [A_2] \wedge f_1 \wedge f_2 \wedge f_d(V_1 \cup V_2 \cup X_d)$$

Alors, le langage $L(\mathcal{C}_1) \cdot L(\mathcal{C}_2)$ est accepté par le CQDD:

$$\mathcal{C}_1 \cdot \mathcal{C}_2 = \{\langle A_d, f_{det} \rangle\}$$

Généralisation de la preuve à n dimensions: Nous pouvons supposer pour chaque dimension que les deux automates A_1 et A_2 ont la forme spéciale demandée. Ensuite, nous construisons une composante d'acceptation en prenant comme contrainte la conjonction des contraintes pour chaque dimension. □

5.3.5.4 Fermeture par dérivation à gauche

Dans cette section nous montrons que les CQDD's sont fermées par dérivation à gauche.

Définition 5.26 :

Soient $\mathcal{L}_1$ et $\mathcal{L}_2$ deux multi-langages à n dimensions. La *dérivée gauche* de $\mathcal{L}_1$ par $\mathcal{L}_2$, noté $\mathcal{L}_2^{-1} \cdot \mathcal{L}_1$ est l'ensemble $\{\vec{w} \in (\Sigma^*)^n \ : \ \exists \vec{w}' \in \mathcal{L}_2. \ \vec{w}'\vec{w} \in \mathcal{L}_1\}$ i.e., l'ensemble de tous les multi-mots qui permettent de d'étendre un élément de $\mathcal{L}_2$ à un élément de $\mathcal{L}_1$.

Étant donné un multi-langage $\mathcal{L}$, nous écrivons $Pref(\mathcal{L})$ pour l'ensemble de préfixes de multi-mots dans $\mathcal{L}$, i.e., $\{\vec{w} \in (\Sigma^*)^n \ : \ \exists \vec{w}' \in (\Sigma^*)^n. \ \vec{w}\vec{w}' \in \mathcal{L}\}$.

Proposition 5.5 :

Pour chaque $n \geq 1$, la classe des CQDD à n dimensions est fermée par dérivation gauche.

Preuve:
Considérons les deux CQDD's $\mathcal{C}_1 = \{\langle A_1, f_1(V_1)\rangle\}$ et $\mathcal{C}_2 = \{\langle A_2, f_2(V_2)\rangle\}$ où, pour $i \in \{1, 2\}$, $A_i = (Q_i, \Sigma, q^i_{init}, \to_i, q^i_{fin})$, et V_i et un sur-ensemble de $X_i = \{x_t \ : \ t \in \to_i\}$. Nous montrons qu'il y a un CQDD qui accepte $L(\mathcal{C}_2)^{-1} \cdot L(\mathcal{C}_1)$. Nous construisons d'abord les automates $\overrightarrow{A_1^q}$ et $\overleftarrow{A_{1,2}^q}$ et l'ensemble F comme dans la construction du Lemme 5.3.

tel-00004890, version 1 - 19 Feb 2004

Nous devrons exprimer le fait que les mots acceptés doivent être des suffixes de mots dans $L(\mathcal{C}_1)$ et des prolongations de mots dans $L(\mathcal{C}_2) \cap Pref(L(\mathcal{C}_1))$. Donc, nous acceptons des éléments de $L(\overrightarrow{A_1^q})$ seulement les mots w (soit ρ_w le calcul accepteur de w dans $\overrightarrow{A_1^q}$) tels qu'il existe un mot w' (le préfixe enlevé) ayant un calcul accepteur $\rho_{w'}$ dans $\overleftarrow{A_{1,2}^q}$ qui satisfait f_2 (i.e., $\rho_{w'}$ satisfait $f_2 \wedge [\overleftarrow{A_{1,2}^q}]$), et tel que $\rho_{w'}\rho_w$ (qui est un calcul accepteur de $w'w$ dans A_1) satisfait la formule f_1. La seule difficulté vient du fait que les transitions du calcul accepteur pour ww' dans A_1 qui sont dans le circuit avec l'état q peuvent faire partie de $\overleftarrow{A_{1,2}^q}$ où de $\overrightarrow{A_1^q}$).

Pour exprimer les contraintes nous avons besoin de renommer les variables associées aux transitions de A_1 et A_2.

Soit $Y_i = \{y_t \ : \ t \in \rightarrow_i\}$ et soient $U_i = (V_i \setminus X_i) \cup Y_i$, pour $i \in \{1, 2\}$. Alors, soient

$X_q = \{x_t \ : \ t \in \mathcal{T}(\overrightarrow{A_1^q})$ et t est dans le circuit contenant $q\}$

et

$X'_q = \{x_t \ : \ t \in \mathcal{T}(\overrightarrow{A_1^q})\}$.

Le langage $L(\mathcal{C}_2)^{-1} \cdot L(\mathcal{C}_1)$ est accepté par le CQDD:

$$\mathcal{C}_2^{-1} \cdot \mathcal{C}_1 = \{\langle \overrightarrow{A_1^q}, f_{deriv}^q((X'_q \setminus X_q) \cup U_1 \cup U_2 \cup X_{1,2})\rangle \ : \ q \in F\} \tag{5.8}$$

où $X_{1,2} = \{x_{\langle t_1, t_2\rangle} \ : \ t_1 \in \rightarrow_1 \ \text{ et } t_2 \in \rightarrow_2\}$ et la formule f_{deriv}^q est donnée par:

$$f'_1 \wedge f'_2 \wedge [\overleftarrow{A_{1,2}^q}] \wedge \bigwedge_{t_1 \in \rightarrow_1} y_{t_1} = \sum_{t_2 \in \rightarrow_2} x_{\langle t_1, t_2\rangle} \wedge \bigwedge_{t_2 \in \rightarrow_2} y_{t_2} = \sum_{t_1 \in \rightarrow_1} x_{\langle t_1, t_2\rangle} \tag{5.9}$$

où les formules f'_1 et f'_2 sont définies par:

$$\begin{aligned} f'_1 &= f_1[y_t/x_t \ : \ x_t \in X_1 \setminus X'_q][(y_t + x_t)/x_t \ : \ x_t \in X_q] \\ f'_2 &= f_2[y_t/x_t \ : \ x_t \in X_2] \end{aligned}$$

Généralisation de la preuve à n dimensions:

Nous construisons d'abord les automates $\overrightarrow{A_1^q}$ et $\overleftarrow{A_{1,2}^q}$ pour chaque dimension. Pour chaque combinaison de ces automates nous obtenons une composante d'acceptation où les contraintes sont modifiées simultanément pour chaque dimension.

□

5.3.5.5 Problèmes du vide, d'appartenance et d'inclusion

Nous montrons que le problème du vide est décidable pour toute la classe CSA.

Proposition 5.6 :

Le problème du vide est décidable pour CSA's.

tel-00004890, version 1 - 19 Feb 2004

Preuve:
Soit $\mathcal{C} = \{\langle (A_1, \ldots, A_n), f \rangle\}$ un CSA. Clairement, $L(\mathcal{C}) \neq \emptyset$ si et seulement si la formule de Presburger $[A_1] \wedge \cdots \wedge [A_n] \wedge f$ est satisfaisable.

□

Avec les propositions 5.2, 5.3, et 5.6, nous montrons le corollaire suivant:

Corollaire 5.1 :

Pour chaque $n \geq 1$, le problème d'appartenance ainsi que le problème d'inclusion sont décidables pour CQDD's à n dimensions.

5.4 Représentation et manipulation d'ensembles de configurations

Dans cette section nous montrons comment les CQDD's peuvent être utilisés pour représenter des ensembles de configurations d'automates communicants. Nous montrons que la classe d'ensemble de configurations CQDD représentables est fermée par union, intersection et la fonction de successeur *post*. En plus, les problèmes du vide et d'inclusion de cette classe sont décidables. Les CQDD's peuvent donc être utilisés comme une structure de représentation dans une analyse en avant. Nous présentons ensuite le résultat principal de cette section: Les CQDD's sont fermés par $post^*_\theta$, c.-à-d. on peut calculer l'effet d'un nombre quelconque d'itération d'un circuit θ sur un ensemble de configurations CQDD représentables.

Nous montrons aussi, que les CQDD's peuvent aussi être utilisés pour l'analyse en arrière. Pour cela, il suffit de considérer les images miroirs de contenus de files.

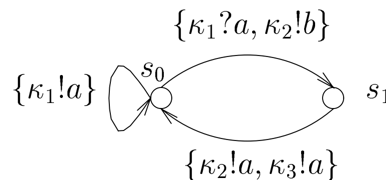
5.4.1 Représentation d'ensemble de configurations

Soit $\mathcal{M} = (S, K, \Sigma, T)$ un CFSM. Chaque ensemble de configurations $C \subseteq Conf$ peut être écrit comme une union $\bigcup_{s \in S} \{s\} \times \mathcal{L}_s$ où les $\mathcal{L}_s$'s sont des multi-langages à $|K|$ dimensions.

Définition 5.27 (CQDD représentable) :

Un ensemble $C \subseteq Conf$ de configurations est CQDD *représentable* (resp. *inverse représentable*) si pour chaque $s \in S$, le multi-langage $\mathcal{L}_s$ est CQDD définissable (resp. inverse définissable)

Considérons comme exemple le système $\mathcal{M}$ représenté par la figure suivante:



tel-00004890, version 1 - 19 Feb 2004

L'ensemble de configuration atteignable à partir de la configuration $(s_0, \epsilon, \epsilon)$ est donné par:

$$\{s_0\} \times \{(a^n, (ba)^m, a^m) \ : \ n, m \geq 0\} \cup \{s_1\} \times \{(a^n, (ba)^m b, a^m) \ : \ n, m \geq 0\} \qquad (5.10)$$

et est clairement CQDD représentable.

5.4.2 Opérations de bases sur les ensembles de configurations

Dans cette section, nous présentons les résultats qui permettent de manipuler et de raisonner sur des ensembles de configurations qui sont CQDD représentables ou inverse CQDD représentable. D'abord avec Propositions 5.1 et 5.6, et Corollaire 5.1 nous déduisons:

Théorème 5.1 :

> La classe des ensembles de configurations CQDD représentable (resp. inverse représentable) est fermée par union, intersection et son problème de vide et d'appartenance est décidable.

Nous montrons ensuite que la classe des ensemble de configurations CQDD représentable est fermé par les opérations *post* (resp. *pre*).

Proposition 5.7 :

> Soit (S, K, Σ, T) une CFSM. Alors, pour chaque $s \in S$, pour chaque $\tau = (s, op, s') \in T$ et pour chaque CQDD $\mathcal{C}$ à $|K|$ dimension, nous pouvons construire un CQDD $\mathcal{C}'$ tel que $\{s'\} \times L(\mathcal{C}') = post_\tau(\{s\} \times L(\mathcal{C}))$.

Preuve:
Soit $\mathcal{I}$ un CQDD avec $L(\mathcal{I}) = \{in(\tau)\}$ et $\mathcal{O}$ un CQDD avec $L(\mathcal{O}) = \{out(\tau)\}$. Alors en utilisant les Propositions 5.5 et 5.4, nous construisons le CQDD $\mathcal{C}' = (\mathcal{I}^{-1} \cdot \mathcal{C}) \cdot \mathcal{O}$. □

Théorème 5.2 :

> Pour chaque ensemble de configuration C, qui est CQDD représentable, l'ensemble de configurations $post(C)$ est CQDD représentable et constructible.

Preuve:
Ce théorème découle immédiatement du fait que les CQDD's sont fermé par union (Proposition 5.1), que la fonction *post* distribue par rapport à l'union, et de la Proposition 5.7. □

Nous obtenons des résultats similaires pour les prédécesseurs immédiats.

Proposition 5.8 :

> Soit (S, K, Σ, T) un CFSM. Alors, pour chaque $s \in S$, pour chaque $\tau = (s', op, s) \in T$ et pour chaque CQDD $\mathcal{C}$ à $|K|$ dimension, nous pouvons construire un CQDD $\mathcal{C}'$ tel que $\{s'\} \times L(\mathcal{C}')^R = pre_\tau(\{s\} \times L(\mathcal{C})^R)$.

tel-00004890, version 1 - 19 Feb 2004

Preuve:
La preuve est symétrique à la preuve de Proposition 5.7. Soit $\mathcal{I}$ un CQDD avec $L(\mathcal{I}) = \{in(\tau)^R\}$ et $\mathcal{O}$ un CQDD avec $L(\mathcal{O}) = \{out(\tau)^R\}$. Alors, $\mathcal{C}' = (\mathcal{O}^{-1} \cdot \mathcal{C}) \cdot \mathcal{I}$. □

Théorème 5.3 :

> Pour chaque ensemble de configuration C, qui est CQDD inverse représentable, l'ensemble de configurations $pre(C)$ est inverse CQDD représentable et constructible.

Preuve:
Ce théorème découle immédiatement du fait que les CQDD sont fermé par union, que la fonction *pre* distribue par rapport à l'union et de Proposition 5.7. □

5.4.3 Calcul de l'effet d'un circuit

Dans cette section nous montrons le résultat technique principal du chapitre. Nous prouvons que les CQDDs sont fermés par l'opération $post^*_\theta$.

Théorème 5.4 :

> Pour chaque ensemble de configuration C, qui est CQDD représentable et chaque circuit θ, l'ensemble de configurations $post^*_\theta(C)$ est CQDD représentable et constructible.

Preuve:
Nous donnons ici les grandes lignes de la preuve. Tous les détails de la preuve se trouvent dans l'annexe A.

Soit (S, K, Σ, T) un CFSM. Nous montrons que pour tout $s \in S$, pour tout circuit θ qui commence avec s et pour tout CQDD $\mathcal{C}$ à $|K|$ dimensions, nous pouvons construire un CQDD $\mathcal{C}'$ avec $\{s\} \times L(\mathcal{C}') = post^*_\theta(\{s\} \times L(\mathcal{C}))$.

Puisque *post* distribue par rapport à l'union, il suffit de considérer des CQDD's $\mathcal{C}$ qui consistent en une seule composante d'acceptation $\langle \mathcal{A}, f \rangle$. Soit $\mathcal{A} = (A_1, \ldots, A_{|K|})$.

Le schéma de la preuve est le suivant:

- Nous construisons pour chaque file indépendamment les successeurs possibles après un certain nombre d'exécutions de θ. Ce nombre est représenté par une variable x_θ. Pour cela nous ne considérons pas les contraintes imposées par f sur les transitions des automates $A_1, \ldots, A_{|K|}$ (ces contraintes sont introduites plus tard). Pour chaque $i \in \{1, \ldots, |K|\}$, nous calculons l'effet de θ après x_θ itérations comme si le système n'avait qu'une seule file κ_i et les configurations étaient données par $L(A_i)$. Cela nous donne un CQDD $\mathcal{D}_i$ à une dimension tel qu'un de ses éléments est un couple $\langle B_i, g_i \rangle$.

tel-00004890, version 1 - 19 Feb 2004

La formule g_i contient comme variables libres x_θ et les variables qui correspondent aux transitions des automate A_i et B_i. Par conséquent, la formule g_i établit un lien entre les contenus initiaux et finaux et tiens compte du fait que ces contenus finaux sont obtenus en exécutant le circuit x_θ fois.

- Le résultat global est calculé ensuite de la manière suivante: Le CQDD qui représente le résultat est une union finie de composantes d'acceptation construites comme suit: Nous prenons de chaque CQDD $\mathcal{D}_i$ un couple $\langle B_i, g_i \rangle$ et nous construisons une composante d'acceptation à $|K|$ dimensions $\langle (B_1, \ldots, B_{|K|}), h \rangle$, où $h = \bigwedge_{i=1}^{|K|} g_i \wedge f$. La formule h fait un lien entre les contenus initiaux et finaux et impose que les contenus initiaux satisfont la formule f. En plus, la formule h fait un lien entre les contenus finaux des files, puisque chaque formule g_i dépend de la *même* variable x_θ, c'est-à-dire ces contenus correspondent au même nombre d'itérations du circuit θ.

La partie essentielle de la preuve est donc de montrer que l'effet d'un circuit sur chaque file peut être construit comme un CQDD à une dimension. La preuve utilise une idée de Jeron [Jér91], qui donne une condition suffisante pour qu'une suite de transitions puisse être répétée perpétuellement. Finkel et Marcé [FM96] ont obtenu indépendemment de nous une partie des résultats présentés dans l'annexe A.

Fixons un circuit θ et une file κ_i et soit $out = out(\theta)$ la sortie du système et $in = in(\theta)$ l'entrée du système par θ. La preuve consiste en une analyse de l'effet de θ sur le contenu de κ. Cette analyse dépend de out, in et la forme du contenu initial. Cette analyse est effectuée d'une manière générique puisque l'effet ne doit pas être calculé seulement pour un seul contenu mais pour un nombre infini de contenus.

Les cas $in = \epsilon$ ou $out = \epsilon$

Commençons cette analyse avec les deux cas simples, où l'un des mots in ou out est vide.

- $in = \epsilon$:
  En commençant avec une configuration w l'effet de répéter le circuit θ x_θ fois est de concaténer le mot out^{x_θ} à droite. Cela donne $w.out^{x_\theta}$. Ils est évident que cet effet peut être construit par un CQDD en utilisant l'opération de concaténation.

- $out = \epsilon$:
  L'effet de répéter le circuit θ x_θ fois à partir d'un mot w est donné par $(in^{x_\theta})^{-1}.w$. Cet effet peut être construit en utilisant l'opération de dérivation à gauche entre CQDD's.

Le cas $out \neq \epsilon$ et $in \neq \epsilon$

Considérons un contenu de la forme $in^k.x$ où k est un entier positif et x un mot sur Σ. Dans ce cas on voit que θ peut être itéré au moins k fois. Après k itérations, les contenus sont donnés par $x.out^k$.

tel-00004890, version 1 - 19 Feb 2004

La première question est si θ peut être encore exécuté. Pour cela, nous avons nécessairement qu'ou bien in est un préfixe de x ou alors x est un préfixe de in. Sinon le système ne serait pas capable de consommer x. Il suffit de considérer le cas où x est un préfixe de in parce que nous pouvons nous restreindre à des contenus de la forme $in^k.x$ avec $|x| < |in|$. Si à partir de $x.out^k$ le système peut encore exécuter le circuit θ, alors nous savons que le système consomme x et va consommer des symboles du mot out^k (par la gauche), et qu'à chaque fois que in peut être consommé, out est ajouté à droite, etc. La question est combien d'itérations peuvent être effectuées de cette manière. Cela dépend évidemment de in, out et x.

Nous considérons deux cas:

1. $|in| \leq |out|$
   Le problème est de déterminer si θ peut être répété perpétuellement ou seulement un nombre fini de fois et quels sont les contenus atteignables. Nous donnons des conditions sur les mots in, out et x qui caractérisent exactement le cas où le circuit peut être répété perpétuellement. Ses conditions expriment qu'il doit y avoir un mot u tel que $out = c.x.u$ où x est un préfixe de in avec $in = x.c$ et $c.x.u = u.c.x$. En plus, nous montrons qu'en commençant par une configuration $in^k.x$ les configurations atteignables après x_θ itérations de θ sont données par $in^k.x.u^{x_\theta}$. L'effet de θ est donc donné en concaténant le langage $\{u^n | n \geq 1\}$ tel que $n = x_\theta$. Ce langage est clairement CQDD définissable.

   Exemple 5.5 :

   Soient $in = ba$, $out = abab$, and $x = b$. Alors, nous avons $c = a$, $u = ab$ et $c.x.u = abab = u.c.x$. Donc, en commençant par le contenu $in.x = bab$, les contenus atteignables après n itérations de θ sont $in.x.u^n = bab(ab)^n$. En effet, nous avons

$$
\begin{aligned}
in.x &= bab \\
&\rightsquigarrow x.out = babab = in.x.u \\
&\qquad \rightsquigarrow x.u.out = x.out.u = bababab = in.x.u^2 \\
&\qquad\qquad \rightsquigarrow x.u^2.out = x.out.u^2 = babababab = in.x.u^3 \\
&\qquad\qquad\qquad \rightsquigarrow \cdots
\end{aligned}
$$

   Nous montrons aussi que si les conditions mentionnés ci-dessus ne sont pas satisfaites, alors il y a une borne k' qui est définie en fonction de in, out et x, tel que pour chaque contenu de la forme $x.out^k$ le circuit θ peut être exécuté au plus k' fois.

2. $|in| > |out|$
   Dans ce cas, pour un contenu fixe le nombre d'itérations est fini. Néanmoins, nous

tel-00004890, version 1 - 19 Feb 2004

considérons ici des ensembles infinis de configurations et nous devons définir l'ensemble des configurations atteignables à partir de toutes les configurations initiales. La difficulté vient du fait que la sortie du système peut être réutilisé comme entrée pour le système. Nous illustrons ce problème par un exemple.

Exemple 5.6 :

Soient $in = abab$, $out = ba$ et $x = a$. Alors, en commençant de $in^5.x = (abab)^5.a$ nous avons la séquence suivante de configurations atteignables:

$$\begin{aligned}
& in^5.x = (abab)^5.a \\
& \quad \rightsquigarrow in^4.x.out = (abab)^4.a.ba \\
& \quad\quad \rightsquigarrow in^3.x.out^2 = (abab)^3.a.(ba)^2 \\
& \quad\quad\quad \ldots \\
& \quad\quad\quad \rightsquigarrow^* x.out^5 = a.(ba)^5 = (abab)^2.a.ba = in^2.x.out \\
& \quad\quad\quad\quad \ldots \\
& \quad\quad\quad\quad \rightsquigarrow^* x.out^3 = a.(ba)^3 = (abab).a.ba = in.x.out \\
& \quad\quad\quad\quad\quad \rightsquigarrow x.out^2 = ababa = in.x \\
& \quad\quad\quad\quad\quad\quad \rightsquigarrow x.out = aba
\end{aligned}$$

Après 5 itérations, le système consomme in^5 et produit out^5, qui permet d'atteindre la configuration $x.out^5$. En fait, $x.out^5$ est égal à $in^2.x.out$. Cela permet deux itérations supplémentaires vers la configuration $x.out^3$. Puisque cette configuration est égale à $in.x$, le circuit peut être exécuté encore une fois.

Nous prouvons dans l'annexe que sous certaines conditions sur in, out et x, toutes les configurations atteignables à partir de $in^k.x$ qui ont une taille supérieure à une borne k' (définie en fonction de la taille de in et out) ont une certaine forme. Cette forme est $in^p.x.out^q$, où p et q sont reliés avec x_θ et k par des contraintes linéaires. En plus, nous montrons que pour chaque k, il y a un successeur de cette forme qui a une taille inférieure à k'. Donc, nous construisons d'abord l'ensemble de tous les successeurs de la forme définie ci-dessus (cet ensemble est CQDD représentable). Ensuite nous calculons leurs successeurs jusqu'à k' itérations de θ (l'image par $post_\theta^{k'}$). Cela permet de capturer toutes les successeurs qui restent, puisque pour chaque configuration, la taille des successeurs décroît strictement.

Si ces conditions mentionnées ci-dessus sur in, out et x ne sont pas satisfaites, nous pouvons montrer qu'il existe une constante k'' telle que, pour chaque contenu de la forme $x.out^k$, le circuit θ peut être exécuté au plus k'' fois.

□

tel-00004890, version 1 - 19 Feb 2004

Nous pouvons aussi prouver que la classe d'ensemble de configurations CQDD inverse représentable est fermée par la fonction pre^*_θ, pour chaque circuit θ.

Théorème 5.5 :

> Pour chaque ensemble de configuration C, qui est CQDD inverse représentable et chaque circuit θ, l'ensemble de configurations $pre^*_\theta(C)$ est CQDD inverse représentable et constructible.

Preuve:
La preuve est la même que pour le Théorème 5.4 en inversant l'interprétation de $in(\theta)$ et $out(\theta)$ et en utilisant leur image inverse (en suivant le même principe que pour transformer la preuve de Proposition 5.7 vers la preuve de Proposition 5.8). □

5.5 Analyse d'atteignabilité en avant et en arrière

Le problème de vérification de base est de tester si une *mauvaise* configuration ne peut jamais être atteinte à partir d'une configuration initiale. Étant donné un ensemble de configurations initiales I et un ensemble de configuration mauvaise B, le problème peut être formulée de deux façons:

P1. $B \cap post^*(I) = \emptyset$,

P2. $I \cap pre^*(B) = \emptyset$.

La première formulation correspond à une analyse d'atteignabilité en avant de l'espace de configuration, tandis que la deuxième correspond à une analyse en arrière.

Donc, étant donné un ensemble de configurations C, nous aimerions calculer l'ensemble des ces successeurs et prédécesseurs, i.e. $post^*(C)$ et $pre^*(C)$. Par définition, pour $\phi \in \{post, pre\}$, nous avons:

$$\phi^*(C) = \bigcup_{i \geq 0} C_i$$

où

$$\begin{aligned} C_0 &= C \\ C_{i+1} &= C_i \cup \phi(C_i) \quad \text{pour tout } i \geq 0 \end{aligned}$$

Dans le cas $\phi = post$ (resp. $\phi = pre$), si C est CQDD représentable (resp. inverse représentable), avec les Théorème 5.1, 5.2, et 5.3 nous pouvons déduire que tous les C_i's sont CQDD représentables (resp. inverse représentables). Les équations ci-dessus donnent donc directement un *semi-algorithme*[1] pour calculer $\phi^*(C)$ basé sur le calcul itératif des C_i's.

1. *semi* dans le sens qu'il ne s'arrête pas toujours

tel-00004890, version 1 - 19 Feb 2004

Puisque la séquence des C_i's est croissante, leur limite est atteinte si pour un indice i nous avons $C_i = C_{i+1}$. Dans ce cas, le semi-algorithme s'arrête et retourne C_i. Nous pouvons détecter si $C_i = C_{i+1}$ puisque le problème d'inclusion est décidable pour des ensembles de configurations CQDD représentables (resp. inverse représentables) (Théorème 5.1). Si les ensembles des états initiaux respectivement mauvais sont aussi CQDD représentable (resp. inverse représentable), le problème P1 (resp. P2) peut être résolu par Théorème 5.1.

Parce que le problème d'atteignabilité pour les CFSM's est indécidable, en général il n'existe pas d'indice i tel que $C_i = C_{i+1}$. Le semi-algorithme naïf ci-dessus ne s'arrête pas en général.

Nous proposons une méthode pour faire face à ce problème de divergence qui s'inspire des "meta"-transitions de Boigelot et Wolper [BW94]. Cette méthode consiste en une *accélération exacte* du calcul itératif de la limite $\phi^*(C)$. Étant donné un ensemble de circuits Θ du graphe de transitions du système, nous ajoutons à chaque pas de calcul l'ensemble de successeurs (ou prédécesseurs) par chaque circuit dans Θ. Cette opération est *exacte* parce que toutes les configurations ajoutées font partie de $\phi^*(C)$. Chaque circuit $\theta \in \Theta$ peut être considéré comme une meta-transition dans le sens de [BW94].

Nous calculons $\phi^*(C)$ comme limite d'une autre séquence croissante de configurations $(D_i)_{i \geq 0}$ donnée par:

$$\begin{aligned} D_0 &= C \\ D_{i+1} &= D_i \cup \phi(D_i) \cup \bigcup_{\theta \in \Theta} \phi^*_\theta(D_i) \quad \text{pour chaque } i \geq 0 \end{aligned}$$

Évidement nous avons $C_i \subseteq D_i$ pour chaque $i \geq 0$. La *chance* d'atteindre la limite $\phi^*(C)$ dans un nombre de pas fini est plus grand ou égale, si on considère la séquence des D_i's au lieu de la séquence des C_i's. Cette chance augmente avec la taille de Θ.

En utilisant les Théorèmes 5.1, 5.2, 5.3, 5.4, et 5.5, nous obtenons un semi-algorithme qui calcule (s'il termine) l'ensemble des successeurs (resp. prédécesseurs) d'un ensemble de configurations CQDD représentable (resp. inverse représentable). Cet algorithme est donné par:

Atteignabilité (Θ, C):
  $X := C$;
  Répéter
    $Y := X$;
    $X := X \cup \phi(X) \cup \bigcup_{\theta \in \Theta} \phi^*_\theta\ (X)$
  jusqu'à $X = Y$;
  Retourner(X)
Fin Atteignabilité

Plusieurs semi-algorithmes d'atteignabilité peuvent être dérivés de **Atteignabilité** (Θ, C) en déterminant des stratégies adéquates pour choisir l'ensemble de circuits Θ.

tel-00004890, version 1 - 19 Feb 2004

Par exemple, le semi-algorithme d'atteignabilité en avant donné dans [BG96] peut être vu comme une instance particulière de notre algorithme si l'analyse commence avec un ensemble fini d'états initiaux.

5.6 Comparaison avec d'autres travaux existants

Dans cette section nous comparons notre travail avec d'autres travaux sur la vérification de systèmes à file. Dans [QJ96, BQ96, Que97] un semi-algorithme de *model-checking* pour les CFSM est proposé. Ce semi-algorithme essaie d'abord de construire une représentation finie du graphe d'états infini généré par le système. Cette représentation est donnée par une grammaire de graphe. Ensuite, des propriétés exprimées par la logique temporelle CTL peuvent être vérifiés sur cette représentation. Cette approche est différent de la notre, parce qu'elle est basée sur la représentation finie du graphe *infini* d'états du systèmes, tandis que la notre est basée sur la représentation fini d'ensembles d'états infinis. D'autres auteurs [AJ96, CFI96] analysent des CFSM's qui utilisent des files d'un fonctionnement incertain (perte de messages, duplication de messages etc.). Dans [CF97] une classe restreintes de CFSM's est introduite pour qui l'espace d'états atteignables est régulier et constructible. Si le CFSM a deux files la restriction est que dans chaque configuration atteignable au moins une des files doit être vide. Les CFSM's que nous considérons n'ont pas ces restrictions.

Dans [BGWW97] le pouvoir des QDD's de [BG96] est analysé en détail. Des conditions nécessaires et suffisantes sur la forme d'un circuit θ dans le graphe de contrôle sont données, pour que l'image par $post_\theta^*$ d'un ensemble régulier reste un ensemble régulier. Un algorithme pour calculer l'image par $post_\theta^*$ pour ces circuits est donné. Cela permet de considérer plus de circuits que dans [BG96]. Cette classe de circuit reste néanmoins assez restreinte. Notre approche permet de considérer *tout* circuit.

Finalement, Peng et Puroshotham [PP91] proposent des algorithmes (qui terminent toujours) qui calculent des approximations supérieures de l'ensemble de configurations atteignables.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Chapitre 6

Conclusion

Bilan

Nous avons étudié dans cette thèse le problème de la vérification de systèmes infinis. Nous avons considéré la méthode du *model-checking*. Nous avons dans une première partie obtenu des résultats d'indécidabilité et de complexité pour le model-checking de systèmes infinis par rapport à des logiques de spécification propositionnelles (régulières).

Nous avons considéré ensuite le problème de la vérification pour des propriétés non-régulières. En effet, les logiques temporelles propositionnelles classiques bien adaptées pour la spécification de systèmes *finis* ne permettent pas d'exprimer des propriétés importantes de systèmes *infinis*. Nous avons considéré des propriétés non-régulières qui portent sur le nombre d'occurrences d'événements. Nous avons introduit la logique temporelle CLTL qui est une combinaison de la logique temporelle linéaire propositionnelle avec l'arithmétique de Presburger. Nous avons étudié en détail la décidabilité du problème de la vérification de CLTL et de plusieurs de ses fragments pour plusieurs classes de systèmes infinis. Nous avons obtenu des résultats de décidabilité du problème de la vérification de classes significatives de systèmes infinis (réseaux de Petri, automates à pile) pour des logiques qui sont plus expressives que les logiques temporelles propositionnelles classiques.

Nous nous sommes ensuite intéressés aux automates communicants qui ont la puissance d'une machine de Turing. Nous avons appliqué l'approche de l'analyse symbolique. Nous avons introduit des structures finies de représentation de configurations, appelées CQDD, qui permettent de représenter un nombre infini de configurations. Ces structures de représentation combinent les automates finis avec des contraintes arithmétiques linéaires et permettent de représenter des ensembles non-réguliers de configurations. Le résultat principal est que ces structures permettent de calculer l'effet de l'exécution répétée de tout circuit dans le graphe de contrôle de l'automate communicant. Ce résultat généralise les résultats obtenus jusqu'à présent. En utilisant ce résultat nous avons défini un algorithme qui, s'il s'arrête, calcule l'espace de configurations atteignables d'un automate communicant.

La logique CLTL et les CQDD sont tous les deux basés sur une combinaison d'une structure régulière (logique temporelle propositionnelle respectivement automate fini) avec l'arithmétique de Presburger. Cette combinaison permet dans les deux cas d'étendre les

résultats existants et de décrire d'une part des propriétés non-régulières et d'autre part des ensembles de configurations non-régulières.

Perspectives

La théorie de la vérification automatique de systèmes infinis est encore à ses débuts. Puisque ces modèles sont utilisés dans beaucoup de domaines, ils existent une multitude de différents formalismes. Par conséquent, beaucoup de problèmes concernant la vérification automatique restent ouverts.

Propriétés non-régulières

La logique $\text{CLTL}_{\Box}$ pour qui nous avons prouvé des résultats de décidabilité du problème de la vérification a un pouvoir expressif "régulier" maximal et était obtenu en interdisant dans CLTL les formules qui causent l'indécidabilité du problème de vérification. La vérification de formules de la forme $[\vec{x} : \vec{\pi}].\ \Diamond f(\vec{x})$ est indécidable déjà pour les systèmes finis. Autrement dit, le problème de satisfaisabilité relative à un système fini d'une formule de la forme $[\vec{x} : \vec{\pi}].\ \Box g(\vec{x})$ est indécidable. Une manière d'obtenir un fragment plus expressif que $\text{CLTL}_{\Box}$ serait donc d'autoriser des propriétés d'inévitabilité où les formules de Presburger ont une forme restreinte, par exemple en interdisant des comparaison entre deux variables.

Les logiques temporelles linéaires régulières ont une caractérisation par les ω-automates finis. Il serait intéressant d'étudier si la logique CLTL peut être caractérisée par une sous-classe des automates avec contraintes arithmétiques [Pei94] en ajoutant des conditions d'acceptation "à la Büchi".

L'analyse des automates communicants

Pour analyser les automates communicants nous avons donné une méthode d'analyse en calculant l'espace de configurations atteignables avec un semi-algorithme qui est paramétré par un ensemble de circuits qui sont utilisés pour l'accélération du calcul. Une question importante est de savoir quels circuits faut-il choisir pour un certain système pour garantir l'arrêt. Un autre problème est de trouver des sous-classes de systèmes pour qui on peut prouver que le semi-algorithme s'arrête toujours. L'application pratique des résultats se heurtent à la complexité du calcul des successeurs ou prédécesseurs. D'où l'importance de savoir pour quel genre de circuits le calcul des successeurs ou de prédécesseurs peut être simplifié.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Annexe A

Preuve du Théorème 5.4

A.1 Analyse de l'effet d'un circuit

Dans cette section nous analysons l'effet sur le contenu *d'une* file de l'exécution répétée d'un circuit θ dans le graphe de transition d'un automate communicant Nous supposons pour cette analyse que toutes les autres files restent inchangées. Nous considérons le cas où le contenu de la file est tel que le circuit peut être exécuté en consommant des messages au-delà du contenu de départ, c'est-à-dire l'ajout de messages à la fin de file permet de continuer l'exécution du circuit après avoir consommer tous les messages qui y étaient initialement. Tous les autres cas sont faciles à analyser.

Soit κ_i une file.

Définition A.1 :

> Pour chaque file dans l'automate communicant le circuit θ définit un mot in_i (la composante i de $in(\theta)$) et un mot out_i (la composante i de $out(\theta)$).

Nous considérons deux cas:

- La taille du contenu de la file croît ou reste constante après une exécution du circuit et un contenu donné.
- La taille du contenu de la file décroît après une exécution du circuit et un contenu donné.

A.1.1 Analyse d'un circuit avec taille croissante

Dans ce cas, nous considérons in_i et out_i de la forme

$$|out_i| \geq |in_i|, out_i \neq \epsilon \text{ et } in_i \neq \epsilon \quad (A.1)$$

Ici la taille de ce qui est écrit dans la file est supérieure ou égale de ce qui y est lu. Étant donné un contenu fixe de la file, ce contenu croît ou reste de la même taille. Nous montrons

ici qu'il est possible de décider si un circuit peut être exécuté perpétuellement et nous donnons l'effet de son exécution répété sur le contenu de la file. Il est évident que pour exécuter le circuit un nombre infini de fois les contenus de départ doivent avoir la forme $in_i^k.x$ où x est un préfixe de in_i. Soit donc dans ce que suit $in_i = x.c$ avec $x, c \in \Sigma^*$ et $c \neq \epsilon$.

Définition A.2 :

Nous appelons x *répétant* si et seulement si

$$\exists u, c_1, c_2 \in \Sigma^* \; \exists k_1 \geq 0 \text{ avec } out_i = c.x.u, u = (c.x)^{k_1}.c_1 \text{ et } c.x = c_1.c_2 \tag{A.2}$$

et

$$c.x.u = u.c.x \tag{A.3}$$

Lemme A.1 :

Si x est répétant alors $out_i.u = u.out_i$

Preuve:
Nous avons avec (A.3) $out_i.u = c.x.u.u = u.c.x.u = u.out_i$. □

Nous pouvons prouver que les x's qui sont répétants permettent d'exécuter un nombre infini de fois le circuit et si le circuit peut être exécuté un nombre infini de fois, x doit être répétant. Formellement:

Lemme A.2 :

x est répétant si et seulement si à partir d'un contenu $in_i^k.x$ et $k > 0$ le circuit peut être exécuté un nombre infini de fois. Si x est répétant, le contenu après n exécutions est donné par $in_i^k.x.u^n$.

Preuve:

- "$\Rightarrow$": Soit x répétant. Nous prouvons par induction qu'à partir d'un contenu $in_i^k.x$ et $k > 0$ le circuit peut être exécuté un nombre infini de fois et que le contenu après n exécutions est donné par $in_i^k.x.u^n$.
  - Base de l'induction:
    Étant donné $in_i^k.x$, une exécution du circuit donne $in_i^{k-1}.x.out_i$. En utilisant (A.2) c'est égal à $in_i^{k-1}.x.c.x.u$. Puisque $in_i = x.c$ cela donne $in_i^k.x.u$.

tel-00004890, version 1 - 19 Feb 2004

- Pas de l'induction:
  $in_i^k.x$ peut être exécuté n fois en donnant $in_i^k.x.u^n$. Ensuite, une autre exécution donne $in_i^{k-1}.x.u^n.out_i$. Utilisant le Lemme A.1 c'est égal à $in_i^{k-1}.x.out_i.u^n = in_i^{k-1}.x.c.x.u.u^n = in_i^k.x.u^{n+1}$.

– "$\Leftarrow$": À partir d'un contenu $in^k.x$, k exécutions du cycle donne d'abord un contenu $x.out^k$. Ensuite, puisque le circuit peut être exécuté un nombre infini de fois, pour tout $k' \in I\!N$ il existe un $x' \in \Sigma^*$ tel que

$$x.out_i^{k+k'} = in_i^{k'}.x' \tag{A.4}$$

Puisque $in_i = x.c$, (A.4) est équivalent à

$$out_i^{k+k'} = (c.x)^{k'-1}.c.x' \tag{A.5}$$

(A.5) implique

$$\exists u, c_1, c_2 \in \Sigma^* \; \exists k_1 \geq 0 \text{ avec } out_i = c.x.u, u = (c.x)^{k_1}.c_1 \text{ et } c.x = c_1.c_2 \tag{A.6}$$

ce qui est équivalent à la condition (A.2).

En utilisant les équations de (A.2) dans (A.5) nous obtenons

$$(c.x.(c.x)^{k_1}.c_1)^{k+k'} = (c.x)^{k'-1}.c.x' \tag{A.7}$$

pour tout $k' \in I\!N$. Nous avons donc

$$c.x.\overbrace{c.x.\ldots.c.x}^{k_1 \text{ fois}}.c_1.c.x \cdots = c.x.\overbrace{c.x.\ldots.c.x}^{k_1 \text{ fois}}.c.x.c_1.c_2 \cdots$$

Cela implique $c_1.c.x = c.x.c_1$. L'équation

$$c.x.u = u.c.x \tag{A.8}$$

découle facilement.

□

Si la condition (A.6) n'est pas satisfaite, alors en commençant par un contenu $x.out_i^k$ le circuit peut être exécuté au plus $max\{k_3 \;:\; \exists d \in \Sigma^*.\; out_i = (c.x)^{k_3}.d\}$ fois. Si la condition (A.6) est satisfaite, pour obtenir (A.8) nous avons besoin de $k' \geq k_1 + 3$. Pour les x's qui ne sont pas répétants nous obtenons donc le corollaire suivant

Corollaire A.1 :

Si x n'est pas répétant alors, à partir d'un contenu $x.out_i^k$ le circuit peut être exécuté au plus k_2 fois où k_2 est une constante qui ne dépend que de out_i, in_i et x. k_2 est donné par $max\{k_3 + 3 \;:\; \exists d \in \Sigma^* \;:\; out_i = (c.x)^{k_3}.d\}$

tel-00004890, version 1 - 19 Feb 2004

A.1.2 Analyse d'un circuit avec taille décroissante

Ici nous considérons in_i et out_i avec les conditions suivantes:

$$|in_i| > |out_i|, out_i \neq \epsilon \text{ et } in_i \neq \epsilon \tag{A.9}$$

Dans ce cas, si nous commençons à exécuter le circuit à partir d'un contenu fixe, il y a seulement un nombre fini de successeurs possibles, parce que la longueur du contenu de file décroît après chaque pas. Mais parce que nous considérons des ensembles infinis de configurations. Nous avons donc besoin de définir l'ensemble de configurations atteignables pour tous.

Étant donné un contenu $in_i^k.x$, k exécutions du circuit donne le contenu $x.out_i^k$. Nous pouvons montrer qu'il y a une constante k_2 qui ne dépend que de in_i et out_i telle que si nous pouvons exécuter le circuit plus que k_2 fois à parti d'un contenu $x.out_i^k$, alors les contenus doivent avoir une forme spéciale que nous pouvons exprimer par des contraintes linéaires. Pour ces états nous allons donner une caractérisation de tous leurs états successeurs.

Soit pour le reste de cette section $in_i = x.c$ avec $x, c \in \Sigma^*$ et $c \neq \epsilon$.

Définition A.3 :

Nous appelons x *répétant* si et seulement si

$$\exists\, c_1, c_2 \in \Sigma^* \; \exists\, k_1 > 0 \text{ avec } c.x = out_i^{k_1}.c_1 \text{ et } c_1.c_2 = out_i \tag{A.10}$$

et

$$c_1.c_2 = c_2.c_1 \tag{A.11}$$

Lemme A.3 :

Si x est répétant, alors

$$in_i^{|out_i|}.x = x.out_i^{|in_i|} \tag{A.12}$$

Preuve:
(A.11) implique $out_i.c_1 = c_1.out_i$. Par ailleurs pour tout $m \in I\!N$ nous avons $out_i^m = c_1^m.c_2^m$. En posant $m = |out_i|$, on obtient

$$out_i^{|c_1|} = c_1^{|out_i|} \tag{A.13}$$

Donc, $in_i^{|out_i|}.x = (x.c)^{|out_i|}.x = x.(c.x)^{|out_i|} = x.(out_i^{k_1}.c_1)^{|out_i|} = x.out_i^{k_1|out_i|}.c_1^{|out_i|}$
$= x.out_i^{k_1|out_1|+|c_1|} = x.out_i^{|in_i|}$. □

tel-00004890, version 1 - 19 Feb 2004

Un corollaire simple du Lemme A.3:

Corollaire A.2 :

Si x est répétant, alors

$$\forall\, d_1 > 0, in_i^{d_1|out_i|}.x = x.out_i^{d_1|in_i|} \tag{A.14}$$

Lemme A.4 :

x n'est pas *répétant* si et seulement s'il y a une constante $k_2 \in I\!N$ telle qu'à partir d'un contenu $x.out_i^k$ avec $k > 0$ le circuit peut être exécuté au plus $k_2 = max\{k_3 + 3 \,:\, \exists d \in \Sigma^*.\ c.x = out_i^{k_3}.d\}$ fois.

Preuve:

– "⇒": Supposons que la constante n'existe pas. Nous montrons que cela implique que x est répétant. Posons un contenu $x.out_i^k$. Pour chaque k', le circuit peut être exécuté k' fois. Cela implique que pour tout k' il existe un $x' \in \Sigma^*$ avec

$$x.out_i^{k+k'} = in_i^{k'}.x' \tag{A.15}$$

Pour $k' > 0$ (A.15) est équivalent à

$$out_i^{k+k'} = (c.x)^{k'-1}.c.x' \tag{A.16}$$

Pour $k' > 1$, parce que (A.16) et $|in_i| > |out_i|$ nous avons

$$\exists\, c_1, c_2 \in \Sigma^*\ \exists\, k_1 > 0 \text{ avec } c.x = out_i^{k_1}.c_1 \text{ et } c_1.c_2 = out_i \tag{A.17}$$

Cela correspond à la condition (A.10). Utilisant les équations (A.10) dans (A.16) implique

$$(c_1.c_2)^{k+k'} = ((c_1.c_2)^{k_1}.c_1)^{k'-1}.c.x' \tag{A.18}$$

Maintenant, si $k' \geq k_1 + 3$ alors (A.18) implique

$$\overbrace{c_1.c_2 \ldots c_1.c_2}^{k_1 \text{ fois}}.c_1.c_2.c_1 \cdots = \overbrace{c_1.c_2 \ldots c_1.c_2}^{k_1 \text{ fois}}.c_1.c_1.c_2 \cdots \tag{A.19}$$

Il s'en suit que

$$c_1.c_2 = c_2.c_1 \tag{A.20}$$

.

tel-00004890, version 1 - 19 Feb 2004

– "$\Leftarrow$": Supposons que x est répétant. Alors, nous avons

$$\exists\, c_1, c_2 \in \Sigma^* \;\exists\, k_1 > 0 \text{ avec } c.x = out_i^{k_1}.c_1 \text{ et } c_1.c_2 = out_i$$

et $c_1.c_2 = c_2.c_1$. Avec le Corollaire A.2 nous avons

$$\forall\, d_1 > 0, in_i^{d_1|out_i|}.x = x.out_i^{d_1|in_i|}$$

La constante ne peut donc pas exister parce que $d_1|in_i|$ n'est pas borné et l'équation $in_i^{d_1|out_i|}.x = x.out_i^{d_1|in_i|}$ montre que le circuit peut être exécuté.

□

Avec ce lemme et le Corollaire A.2 nous prouvons le lemme suivant. Soit $n_1 = |out_i|$ et $n_2 = |in_i|$.

Lemme A.5 :

Si x est répétant, alors pour des contenus $in_i^k.x$ avec $k \geq 4n_1$, si le contenu du successeur après n' du circuit a une taille $\geq 3n_1n_2 + |x|$, alors il a la forme

$$in^{n_1+d_2-d_4}.x.out^{(d_1-1)n_2-d_3(n_2-n_1)+d_4}$$

où

- $n_1 = |out_i|$ et $n_2 = |in_i|$,
- $k = d_1n_1 + d_2$ avec $0 \leq d_2 < n_1$,
- $n' = d_3n_1 + d_4$ avec $0 < d_4 \leq n_1$,
- $(d_1 \Leftrightarrow 1)n_2 \Leftrightarrow d_3(n_2 \Leftrightarrow n_1) + d_4 \geq 0$

Preuve:
Fixons un k avec $k \geq 4n_1$. Nous montrons le lemme par induction sur n'. Comme base d'induction nous montrons que le lemme est vrai pour tout $n' \leq n_1$. Ensuite, nous montrons que pour tout $d_3 \in I\!N$ si le lemme est vrai pour $n' \leq (d_3+1)n_1$ alors il est vrai pour tout n' avec $(d_3+1)n_1 < n' \leq (d_3+2)n_1$.

– Base d'induction: $1 \leq n' \leq n_1$: Dans ce cas, $d_4 = n'$, $d_3 = 0$ et tous les successeurs de $in^k.x = in^{d_1n_1+d_2}.x$ sont donnés par $in^{d_1n_1+d_2-n'}.x.out^{n'}$. Parce que $n_1 + d_2 \geq n'$ et avec le Corollaire A.2 nous avons

$$in^{d_1n_1+d_2-n'}.x.out^{n'} = in^{n_1+d_2-n'}.x.out^{(d_1-1)n_2+n'}$$

qui a la forme requise.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

– Pas d'induction:
  Étant donné un $d_3 \in I\!N$, soit n' tel que $(d_3+1)n_1 < n' \leq (d_3+2)n_1$. Cela implique que $n' = (d_3+1)n_1 + d_4$ avec $0 < d_4 \leq n_1$. Supposons que in^k a un successeur après n' exécutions avec taille $\geq 3n_1n_2 + |x|$. Alors, le successeur après $n' \Leftrightarrow n_1$ exécutions du circuit a aussi une taille $\geq 3n_1n_2 + |x|$ (la taille diminue après chaque exécution du circuit).

  D'après l'hypothèse d'induction les successeurs après $n' \Leftrightarrow n_1 = d_3n_1 + d_4$ exécutions sont donnés par

$$in_i^{n_1+d_2-d_4}.x.out_i^{(d_1-1)n_2-d_3(n_2-n_1)+d_4}$$

  Puisque la longueur des contenus obtenus est supérieure à $3n_1n_2 + |x| = 2n_1|in_i| + |x| + n_2|out_i|$ nous avons $(d_1 \Leftrightarrow 1)n_2 \Leftrightarrow d_3(n_2 \Leftrightarrow n_1) + d_4 \geq n_2$ et en utilisant le Corollaire A.2 nous avons

$$in^{n_1+d_2-d_4}.x.out^{(d_1-1)n_2-d_3(n_2-n_1)+d_4} = in^{2n_1+d_2-d_4}.x.out^{(d_1-1)n_2-(d_3+1)(n_2-n_1)-n_1+d_4}$$

  Donc, les successeurs après n_1 pas en plus sont donnés par

$$in^{n_1+d_2-d_4}.x.out^{(d_1-1)n_2-(d_3+1)(n_2-n_1)+d_4}$$

□

A.2 Fermeture par $post_\theta^*$

Avec les résultats de la section précédente nous pouvons montrer que la classe des ensembles de configurations CQDD représentables est fermée par la fonction $post_\theta^*$ pour chaque circuit θ dans le graphe de transitions du système.

Théorème 5.4:

> Pour chaque ensemble de configuration C, qui est CQDD représentable et chaque circuit θ, l'ensemble de configurations $post_\theta^*(C)$ est CQDD représentable et constructible.

Pour prouver ce théorème il suffit de prouver la proposition suivante.

Proposition A.1 :

> Soit (S, K, Σ, T) une CFSM. Alors, pour chaque circuit θ à partir d'un $s \in S$, et pour chaque CQDD à $|K|$ dimension $\mathcal{C}$, nous pouvons construire un CQDD $\mathcal{C}'$ avec $\{s\} \times L(\mathcal{C}') = post_\theta^*(\{s\} \times L(\mathcal{C}))$.

tel-00004890, version 1 - 19 Feb 2004

Preuve:
Parce que *post* distribue par rapport à l'union, il suffit de raisonner sur des CQDD's $\mathcal{C}$ qui consistent en une composante d'acceptation $\langle \mathcal{A}, f \rangle$. Soit $\mathcal{A} = (A_1, \dots, A_{|K|})$.

D'abord nous construisons pour chaque file indépendamment tous les états successeurs possibles après n exécution de θ. Pour faire cela nous ne considérons pas la contrainte f sur les transitions de $A_1, \dots, A_{|K|}$, c'est-à-dire que nous calculons les successeurs des états donnés par $L(A_i)$. Cela nous donne un CQDD à une seule dimension pour chaque file. Chaque élément de ce CQDD est un couple $\langle B_i, g_i \rangle$. La formule g_i contient des variables libres qui viennent des transitions de l'automate initiale A_i, des automates intermédiaires utilisés dans la construction, et de B_i. En plus g_i contient comme variable libre x_θ qui représente le nombre d'exécutions d'un circuit.

Les résultats sont connectés en construisant tous les $|K|$-uplets possibles en choisissant un élément de chaque CQDD pour toutes les files et en prenant comme contrainte la conjonction $\bigwedge_{i=1}^{|K|} g_i \wedge f$. Cela impose que x_θ doit avoir la même valeur pour chaque file et que toutes les variables associées aux transitions des automates $\mathcal{A}_i$ satisfont la contrainte f.

Soit κ_i une file quelconque. Pour cette file nous montrons dans ce que suit comment calculer les états successeurs possibles après x_θ exécution du circuit.

Ces états vont être donné par un CQDD à une dimension $\mathcal{D}_i$ qui contient x_θ comme variable libre.

Pour donner cette construction nous utilisons les résultat de la section A.1 et nous avons besoin de quelques définitions.

Définition A.4 :

> Soit $y \in \Sigma^*$ A_y est l'automate simple restreint qui accepte le langage $\{y\}$ et B_{y^*} (resp. C_{y^*}) est l'automate simple qui accepte le langage y^*.

Noter que B_{y^*} et C_{y^*} consistent exactement en un circuit. Soit t_B (resp. t_C) une transition de B_{y^*} (resp. C_{y^*}).

Maintenant pour définir $\mathcal{D}_i$ nous utilisons les opérations (dérivation à gauche, concaténation, produit) définie dans la section 5.3.5. Nous considérons cinq cas:

- **Cas 1**: $out_i = \epsilon$ et $in_i = \epsilon$

  Dans ce cas, il n'y a ni écriture ni lecture dans la file. Le contenu de la file ne change donc pas. D'où:

  $$\mathcal{D}_i = \{\langle A_i, tt \rangle\}$$

- **Cas 2**: $out_i = \epsilon$ and $in_i \neq \epsilon$

  Dans ce cas, il n'y a pas d'écriture dans la file mais il y a lecture de la file. On peut lire in_i de la file, si le contenu de la file commence avec in_i. Ceci peut se répéter plusieurs fois. Donc, pour des contenus de la forme $in_i^k.x \in L(A_i)$ pour un $k \in I\!N$ et $x \in \Sigma^*$, $in_i^{k-x_\theta}.x$ est le nouveau contenu après x_θ lectures si $k \geq x_\theta$. Nous avons donc

  $$\mathcal{D}_i = (\langle B_{in_i^*}, x_{t_B} = x_\theta \rangle)^{-1} \cdot \langle A_i, tt \rangle$$

- **Cas 3**: $in_i = \epsilon$ and $out_i \neq \epsilon$

  Dans ce cas, il n'y a pas de lecture de la file, mais il y a écriture dans la file. On peut toujours écrire dans une file. Par conséquent, pour chaque contenu $y \in L(A_i)$, $y.out_i^{x_\theta}$ est le nouveau contenu après x_θ exécution du circuit. Donc,

  $$\mathcal{D}_i = \langle A_i, tt\rangle \cdot \langle B_{out_i^*}, x_{t_B} = x_\theta\rangle$$

- **Cas 4**: $|out_i| \geq |in_i|, out_i \neq \epsilon$ and $in_i \neq \epsilon$

  Dans ce cas, la taille de ce qui est écrit dans la file est supérieure ou égale de ce qui y est lu. Étant donné un contenu fixe de la file, la taille de ce contenu croît ou reste la même.

  Nous avons analysé une partie de ce cas dans la section A.1. Nous avons donné avec le Lemme A.2 une condition nécessaire et suffisante pour qu'un circuit puisse être exécuté perpétuellement à partir d'un contenu de file.

  Cas 4.1

  D'abord nous traitons le cas pour les mots dans $L(A_i)$ qui ont la forme $in_i^k.x$, où x n'est pas un préfixe de in_i et in_i n'est pas un préfixe de x. Dans ce cas le circuit peut être exécuté au plus k fois. Les contenus atteignables à partir de ces mots sont donc donnés par $in_i^{k-x_\theta}.x.out_i^{x_\theta}$ si $k \geq x_\theta$. Par conséquent, $\mathcal{D}_i$ est donné par

  $$\mathcal{D}_i = (((\langle B_{in_i^*}, x_{t_B} = x_\theta\rangle)^{-1} \cdot \langle A_i, tt\rangle) \cdot \langle C_{out_i^*}, x_{t_C} = x_\theta\rangle$$

  $L(\mathcal{D}_i)$ contient aussi des successeurs des contenus traités dans le cas suivant. Mais il est évident que tous les mots dans $L(\mathcal{D}_i)$ sont des successeurs.

  Cas 4.2:

  Nous considérons ici des mots dans $L(A_i)$ qui ont la forme $in_i^k.x$, où x est un préfixe (peut être ϵ) de in_i avec $|x| < |in_i|$. Dans ce cas, le circuit peut être exécutée au moins k fois. Après cela, la forme de out_i détermine si le circuit peut encore être exécuté. Soit $c \in \Sigma^*$ telle que $in_i = x.c$. L'exécution du circuit k fois donne des contenus de la forme $x.out_i^k$, et le circuit pourrait toujours être exécuté.

  Cas 4.2.1 x est répétant:

  Utilisant le Lemme A.2 nous voyons qu'à partir de $in_i^k.x = (x.c)^k.x$ pour $k > 0$, le circuit peut être exécuté un nombre infini de fois et pour x_θ exécutions cela donne les contenus $in_i^k.x.u^{x_\theta}$. A partir d'un contenu x nous pouvons exécuter le circuit infiniment souvent, si nous pouvons l'exécuter une fois. Cela peut être facilement testée et dépend de l'ordre de l'ajout et de suppression de messages du circuit.

  Construction de $\mathcal{D}_i$:

tel-00004890, version 1 - 19 Feb 2004

$\mathcal{D}_i$ est donné par l'union finie de CQDD's $\mathcal{E}_x$ où x est un préfixe de in_i. $\mathcal{E}_x$ est construit de la façon suivante:

Si $u = \epsilon$ alors

$$\mathcal{D}_i = \{\langle A_i, tt\rangle\}$$

Sinon, si le circuit peut être exécuté une fois à partir de x, alors

$$\mathcal{E}_x = (((\langle B_{in_i^*}, tt\rangle \cdot \langle A_x, tt\rangle) \times \langle A_i, tt\rangle) \cdot \langle C_{u^*}, x_{t_C} = x_\theta\rangle$$

sinon

$$\mathcal{E}_x = (((\langle B_{in_i^*}, tt\rangle \cdot \langle A_{in_i.x}, tt\rangle) \times \langle A_i, tt\rangle) \cdot \langle C_{u^*}, x_{t_C} = x_\theta\rangle$$

Cas 4.2.2 x n'est pas répétant:

Après n' exécution l'ensemble de contenus atteignables est donné par $in_i^{k-n'}.x.out_i^{n'}$ avec $k \geq n'$. Par le Corollaire A.1 il y a une constante k_2 telle qu'à partir d'un contenu de la forme $x.out_i^{n'}$ le circuit peut être exécuté au plus k_2 fois. Pour capturer tous les successeurs possibles de ces contenus il suffit de calculer les successeurs de $in_i^{k-n'}.x.out_i^{n'}$ avec $k \geq n'$ après au plus k_2 exécutions du circuit.

Construction de $\mathcal{D}_i$:

$\mathcal{D}_i$ est donné par une union finie de CQDD's $\mathcal{E}_x$ où x est un préfixe de in_i. $\mathcal{E}_x$ est construit de la façon suivante:

D'abord soit

$$\mathcal{F}_x = ((((\langle B_{in_i^*}, x_{t_B} = n'\rangle) \cdot \langle A_x, tt\rangle)^{-1} \cdot \langle A_i, tt\rangle) \cdot \langle C_{out_i^*}, x_{t_C} = n'\rangle$$

Cela nous donne tous les successeurs de la forme $in_i^{k-n'}.x.out_i^{n'}$. Si $n' = x_\theta$ cela nous donne toutes les successeurs déjà générés dans le cas 4.1.

Pour obtenir tous les successeurs jusqu'à k_2 exécutions du circuit à partir des états de $\mathcal{F}_x$ nous construisons pour chaque $j \in I\!N$ avec $0 \leq j \leq k_2$, le CQDD $\mathcal{F}_x^j$ tel que $\{s\} \times L(\mathcal{F}_x^j) = post_\theta^j(\{s\} \times L(\mathcal{F}_x))$. Cela est possible, puisque les ensembles CQDD représentables sont fermés par l'opération $post_\tau$ pour chaque transition (voir Théorème 5.2).

tel-00004890, version 1 - 19 Feb 2004

Remarquez que $\mathcal{F}_x = \mathcal{F}_x^0$. Nous devons tenir compte du nombre global x_θ d'exécutions du circuit. Ce nombre est donné par $x_\theta = n' + j$. Par conséquent,

$$\mathcal{E}_x = \bigcup_{j=0}^{k_3} add(\mathcal{F}_x^j, x_\theta = n' + j)$$

où $add(\mathcal{F}_x^j, x_\theta = n' + j)$ remplace chaque formule f qui apparaît dans le CQDD $\mathcal{F}_x^j$ vers $f \wedge x_\theta = n' + j$.

- **Cas 5**: $|in_i| > |out_i|, out_i \neq \epsilon$ and $in_i \neq \epsilon$

  Pour ce cas nous utilisons les résultat de section A.1. Il y a deux cas à considérer.

  Cas 5.1

  Les contenus ont la forme $in_i^k.x$, où x n'est pas un préfixe de in_i et in_i n'est pas un préfixe de x. Alors, l'ensemble de contenus atteignables est donné par $in_i^{k-x_\theta}.x.out_i^{x_\theta}$ avec $k \geq n$. $\mathcal{D}_i$ est donc calculé comme dans cas 4.1.

  Cas 5.2

  Les contenus ont la forme $in_i^k.x$, où x est un préfixe de in_i (peut être ϵ) avec $|x| < |in_i|$. Soit $c \in \Sigma^*$ avec $in_i = x.c$.

  Cas 5.2.1 x est répétant:

  Nous utilisons le Lemme A.5. Soit $n_1 = |out_i|$ et $n_2 = |in_i|$. Pour un contenu $in_i^k.x$ avec $k \geq 4n_1$, l'ensemble des successeurs après n' exécutions du circuit qui sont de longueur $\geq 3n_1n_2 + |x|$ est donné par

$$in^{n_1+d_2-d_4}.x.out^{(d_1-1)n_2-d_3(n_2-n_1)+d_4} \tag{A.21}$$

  où

  - $n_1 = |out_i|$ et $n_2 = in_i$,
  - $k = d_1n_1 + d_2$ avec $0 \leq d_2 < n_1$,
  - $n' = d_3n_1 + d_4$ avec $0 < d_4 \leq n_1$,
  - $(d_1 \Leftrightarrow 1)n_2 \Leftrightarrow d_3(n_2 \Leftrightarrow n_1) + d_4 \geq 0$

  Il est clair que chaque $in_i^k.x$ avec $k \geq 4n_1$ a un successeur u_k de taille inférieure à $4n_1|in_i|$ qui est de la forme (A.21). Nous rappelons qu'à chaque pas la taille décroît par $n_2 \Leftrightarrow n_1$. Donc, il doit y avoir un successeur avec une taille entre $3n_1n_2 + |x|$ et $4n_1n_2 + |x|$). Pour capturer les successeurs de $in_i^k.x$ qui restent, il suffit de calculer les successeurs de contenus de la forme (A.21) après $4n_1n_2 + |x|$ exécutions du circuit. En effet, pour chaque k nous capturons bien tous les successeurs de u_k parce que la taille de chaque configuration décroît après chaque exécution de θ.

tel-00004890, version 1 - 19 Feb 2004

Construction de $\mathcal{D}_i$:
$\mathcal{D}_i$ est donné par l'union finie de CQDD's $\mathcal{E}_x$ où x et un préfixe de in_i. $\mathcal{E}_x$ est construit comme suit:

Nous construisons une formule pour calculer toutes les valeurs possibles de k dans les états de la forme $in_i^k.x$. Soit $\mathcal{F}_x = (\langle B_{in_i^*}, x_{t_B} = k\rangle \cdot \langle A_x, tt\rangle) \times \langle A_i, tt\rangle$. Maintenant, soit f_x la disjonction de toutes les formules de Presburger de chaque élément de $\mathcal{F}_x$. Intuitivement, f_x nous donne des contraintes sur k.

Pour calculer l'ensemble des états successeurs soient g_x la formule de Presburger

$$\begin{array}{ll} f_x & \wedge\ n' = d_3 n_1 + d_4 \ \wedge\ k = d_1 n_1 + d_2 \ \wedge\ 0 < d_4 \leq n_1 \\ & \wedge\ 0 \leq d_2 < n_1 \ \wedge\ (d_1 \Leftrightarrow 1) n_2 \Leftrightarrow d_3 (n_2 \Leftrightarrow n_1) + d_4 \geq 0 \end{array}$$

Soit

$$\begin{array}{ll} \mathcal{G}_x = & \langle B'_{in_i^*}, x_{t_{B'}} = n_1 + d_2 \Leftrightarrow d_4\rangle \cdot \langle A_x, tt\rangle \cdot \\ & \langle C_{out_i^*}, x_{t_C} = (d_1 \Leftrightarrow 1) n_2 \Leftrightarrow d_3 (n_2 \Leftrightarrow n_1) + d_4 \wedge g_x\rangle \end{array}$$

Cela nous donne toutes les successeurs de la forme (A.21). Soit $k_2 = 4 n_1 n_2 + |x|$. Pour obtenir les successeurs jusqu'à k_2 exécutions du circuit à partir des configurations de $\mathcal{G}_x$ nous procédons comme dans le cas 4.2.2 et obtenons $\mathcal{E}_x$.

Cas 5.2.2 x n'est pas répétant:

Similaire au cas 4.2.2.

Finalement, nous pouvons donner le CQDD $\mathcal{C}'$ qui décrit toutes les états successeurs. $\mathcal{C}'$ est le plus petit ensemble tel que

$$\forall \langle B_1, g_1\rangle \in \mathcal{D}_1 \cdots \forall \langle B_{|K|}, g_{|K|}\rangle \in \mathcal{D}_{|K|}.\ \langle (A_1, \ldots, A_{|K|}), \bigwedge_{i=1}^{|K|} g_i \wedge f\rangle \in \mathcal{C}'$$

□

tel-00004890, version 1 - 19 Feb 2004

Bibliographie

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science*, 138, 1995.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 25 April 1994. Fundamental Study.

[AH89] R. Alur and T. A. Henzinger. A really temporal logic. In *Focs 89*, pages 164–169, 1989.

[AJ96] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127:91–101, 1996.

[Ake78] S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(8):261–264, August 1978.

[BBK87] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of Bisimulation Equivalence for Processes Generating Context-Free Languages. Tech. Rep. CS-R8632, 1987. CWI.

[BE97] Burkart and Esparza. More infinite results. *BEATCS: Bulletin of the European Association for Theoretical Computer Science*, 62, 1997.

[BEH95] A. Bouajjani, R. Echahed, and P. Habermehl. On the Verification Problem of Nonregular Properties for Nonregular Processes. In *LICS'95*. IEEE, 1995.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. LNCS 1243, 1997.

[BG96] B. Boigelot and P. Godefroid. Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs. In *CAV'96*. LNCS 1102, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Static Analysis Symposium*, 1997.

tel-00004890, version 1 - 19 Feb 2004

[BH96] A. Bouajjani and P. Habermehl. Constraint properties, semilinear systems and petri nets. In *CONCUR'96*. LNCS 1119, Springer Verlag, 1996.

[BH97] A. Bouajjani and P. Habermehl. Symbolic Reachabilty Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations. In *ICALP*, 1997.

[BK88] J.A. Bergstra and J.W. Klop. Process Theory based on Bisimulation Semantics. In *REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency.* Springer-Verlag, 1988. LNCS 354.

[BM96] A. Bouajjani and O. Maler. Reachability Analysis of Pushdown Automata. In *Infinity'96.* tech. rep. MIP-9614, Univ. Passau, 1996.

[Boc78] G.V. Bochmann. Finite State Description of Communication Protocols. *Computer Networks*, 2, October 1978.

[BQ96] O. Burkart and Y.M. Quemener. Model-Checking of Infinite Graphs Defined by Graph Grammars. In *Infinity'96.* tech. rep. MIP-9614, Univ. Passau, 1996.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[BS97] O. Burkart and B. Steffen. Model checking the full modal mu-calculus for infinite sequential processes. *Lecture Notes in Computer Science*, 1256:419–429, 1997.

[BT76] I. Borosh and L. Treybis. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Amer. Math Soc.*, 55:299–304, 1976.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18. Cambridge Tracts of Computer Science, 1990.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *CAV'94*. LNCS 818, 1994.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 2(5):323–342, 1983.

[CCI88] CCITT. *Recommendation Z.100: Specification and Description Language SDL*, blue book, volume x.1 edition, 1988.

[CE81] E. M. Clarke and E. A. Emerson. Synthesis of Synchronization Skeletons for Branching Time Temporal Logic. In *Logics of Programs: Workshop*, volume 131 of *Lecture Notes in Computer Science*, Yorktown Heights, New York, May 1981. Springer-Verlag.

tel-00004890, version 1 - 19 Feb 2004

[CES83] E.M. Clarke, E.A. Emerson, and E. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications: A Practical Approach. In *10th ACM Symp. on Principles of Programming Languages.* ACM, 1983. Complete version published in ACM TOPLAS, 8(2):244–263, April 1986.

[CF97] G. Cécé and A. Finkel. Programs with quasi-stable channels are effectively recognizable. In *Computer Aided Verification*, volume 1254. LNCS, Springer Verlag, 1997.

[CFI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 10 January 1996.

[CG77a] Rina S. Cohen and Arie Y. Gold. Theory of ω-languages. I: Characterizations of ω-context-free languages. *Journal of Computer and System Sciences*, 15(2):169–184, October 1977.

[CG77b] Rina S. Cohen and Arie Y. Gold. Theory of ω-languages. II: A study of various models of ω-type generation and recognition. *Journal of Computer and System Sciences*, 15(2):185–208, October 1977.

[CG78] Rina S. Cohen and Arie Y. Gold. ω-computations of deterministic pushdown machines. *Journal of Computer and System Science*, 16:275–300, 1978.

[Chr93] Søren Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, University of Edinburgh, 1993.

[Dam92] M. Dam. Fixed points of Büchi automata. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 39–50. Springer-Verlag, 1992.

[Esp94] J. Esparza. On the decidability of model checking for several mu-calculi and Petri nets. In *CAAP: Colloquium on Trees in Algebra and Programming.* LNCS 787, Springer-Verlag, 1994.

[Esp96] J. Esparza. More infinite results. In Steffen and Margaria, editors, *Infinity: International Workshop on Verification of Infinite State Systems.* Tech. Report MIP-9614, Univ. Passau, 1996.

[Esp97] Javier Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.

[FM96] A. Finkel and O. Marcé. Verification of infinite regular communicating automata. Rapport interne, LSV, ENS-Cachan, 1996.

tel-00004890, version 1 - 19 Feb 2004

[FWW97] A. Finkel, B. Willems, and P. Wolper. A Direct Symbolic Approach to Model Checking Pushdown Systems. In *Infinity'97*. tech. rep. Univ. Uppsala, 1997.

[Hab97] Peter Habermehl. On the complexity of the linear-time μ-calculus for Petri Nets. In *Proceedings of the 18th Conference on Application and Theory of Petri Nets*, volume 1248 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[Hab98] Peter Habermehl. *Vérification de systèmes infinis*. PhD thesis, ce document, 1998.

[Har78] M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Pub. Comp., 1978.

[HP79] John Hopcroft and Jean-Jaques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, April 1979.

[HPS83] D. Harel, A. Pnueli, and J. Stavi. Propositional Dynamic Logic of Nonregular Programs. *Journal of Computer and System Sciences*, 26, 1983.

[HRY91] Rodney R. Howell, Louis E. Rosier, and Hsu-Chun Yen. A taxonomy of fairness and temporal logic problems for Petri nets. *Theoretical Computer Science*, 82(2):341–372, 31 May 1991.

[ISO89] ISO. ESTELLE: A formal description technique based on an extended state transition model. Technical Report 9074, %I ISO, 1989. 1989.

[Jan90] P. Jancar. Decidability of a Temporal Logic Problem for Petri Nets. *T.C.S.*, 74:71–93, 1990.

[Jér91] Thierry Jéron. *Contribution à la validation des protocoles: test d'infinitude et vérification à la volée*. PhD thesis, Université de Rennes 1, 1991.

[Kos82] S.R. Kosaraju. Decidability and Reachability in Vector Addition Systems. In *STOC'82*. ACM, 1982.

[Koz83] D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Kur94] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton Univ. Press, 1994.

[Lip76] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.

tel-00004890, version 1 - 19 Feb 2004

[May81] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computation*, pages 238–246, Milwaukee, Wisconsin, 11–13 May 1981.

[May98] Richard Mayr. *Decidability and Complexity of Model-checking Problems for Infinite-state Systems.* PhD thesis, Technical University Munich, 1998. à paraître.

[McM93] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[ME96] S. Melzer and J. Esparza. Checking system properties via integer programming. In Hanne Riis Nielson, editor, *Proceedings of the 6th European Symposium on Programming (ESOP'96)*, volume 1058 of *LNCS*, pages 250–264, Berlin, April 22-24 1996. Springer.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall Int., 1989.

[Mol96] Faron Moller. Infinite results. In *CONCUR: 7th International Conference on Concurrency Theory*, volume 1119. LNCS, Springer-Verlag, 1996.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Par66] R.J. Parikh. On Context-Free Languages. *Journal of the A.C.M.*, 13, 1966.

[Par81] D. Park. Concurrency and Automata on Infinite Sequences. In *5th GI-Conference on Theoretical Computer Science*. 1981. LNCS 104.

[Pei94] Marco Veloso Peixoto. *Automates à Contraintes Arithmetiques et Procédures d'Evaluation Ascendante de Programmes Logiques.* PhD thesis, Université Paris 7, 1994.

[Pet62] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Darmstadt, 1962.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *FOCS'77*. IEEE, 1977.

[PP91] W. Peng and S. Puroshothaman. Data flow analysis of communicating finite state machines. *ACM Transactions on Programming Languages and Systems*, 13(3):399–442, July 1991.

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du permier Congrès des Mathématiciens des Pays slaves*, Warszawa, 1929.

tel-00004890, version 1 - 19 Feb 2004

[QJ96] Y.M. Quemener and T. Jéron. Finitely Representing Infinite Reachability Graphs of CFSMs with Graph Grammars. In *FORTE/PSTV'96*. Chapman and Hall, 1996.

[QS82] J-P. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *Intern. Symp. on Programming, LNCS 137*, 1982.

[Que97] Y.M Quemener. *Vérification de protocoles à espace d'états infini représentable par une grammaire de graphes*. PhD thesis, Université de Rennes I, 1997.

[Rac78] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, April 1978.

[Reu89] C. Reutenauer. *Aspects mathématiques des réseaux de Petri*. Masson, Paris, 1989.

[Rog87] Hartley Rogers Jr. *Theory of Recursive Functions and Effective Computability*. MIT press, Cambridge, MA, 1987.

[RY86] Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *Journal of Computer and System Sciences*, 32(1):105–135, February 1986.

[Saf88] S. Safra. On the complexity of ω-automata. In *29th Annual Symposium on Foundations of Computer Science*, pages 319–327, White Plains, New York, 24–26 October 1988. IEEE.

[Sav70] W. J. Savitch. Relational between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[Tho79] W. Thomas. Star-Free Regular Sets of ω-Sequences. *Information and Control*, 42, 1979.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990.

[Var88] M. Y. Vardi. A temporal fixpoint calculus. In ACM-SIGPLAN ACM-SIGACT, editor, *Conference Record of the 15th Annual ACM Symposium on Principels of Programming Languages (POPL '88)*, pages 250–259, San Diego, CA, USA, January 1988. ACM Press.

[Var96] M. Y. Vardi. An automata-theoretic approach to linear temporal logic (BANFF'94). *Lecture Notes in Computer Science*, 1043, 1996.

[VJ85] R. Valk and M. Jantzen. The Residue of Vector Sets with Applications to Decidability Problems in Petri Nets. *Acta Informatica*, 21, 1985.

tel-00004890, version 1 - 19 Feb 2004

[VW86] M.Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *1st Symp. on Logic in Computer Science*. IEEE, 1986.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 15 November 1994.

[Wal96] Igor Walukiewicz. Pushdown processes: games and model checking. In *CAV'96*. LNCS 836, 1996.

[Wol83] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56, 1983.

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

tel-00004890, version 1 - 19 Feb 2004

Résumé: Cette thèse traite du problème de la vérification de systèmes ayant un nombre infini d'états. Ces systèmes peuvent être décrits par plusieurs formalismes tels que des algèbres de processus ou des automates finis munis de structures de données non-bornées (automates à pile, réseaux de Petri ou systèmes à files).

Dans une première partie de la thèse nous nous intéressons à la caractérisation de classes de systèmes infinis et de propriétés pour lesquels le problème de vérification est décidable. Nous considérons d'abord la complexité de la vérification du mu-calcul linéaire pour les réseaux de Petri. Ensuite, nous définissons des logiques temporelles qui permettent d'exprimer des propriétés non-régulières comportant des contraintes linéaires sur le nombre d'occurrences d'événements. Ces logiques sont plus expressives que les logiques utilisées dans le domaine. Nous montrons en particulier que le problème de la vérification d'une logique qui est plus expressive que le mu-calcul linéaire est décidable pour des classes de systèmes infinis telles que les automates à pile et les réseaux de Petri.

Une deuxième partie de la thèse est consacrée aux systèmes communicant par files d'attente, dont le problème de vérification est en général indécidable. Nous appliquons le principe de l'analyse symbolique à ces systèmes. Nous proposons des structures finies qui permettent de représenter et de manipuler des ensembles infinis de configurations de tels systèmes. Ces structures permettent de calculer l'effet exact d'une exécution répétée de tout circuit dans le graphe de transitions du système. Ainsi, chaque circuit peut être considéré comme une nouvelle "transition" du système. Nous utilisons ce résultat pour accélérer le calcul de l'ensemble des configurations atteignables d'un système afin d'augmenter les chances de terminaison de ce calcul.
Mots-clés: vérification automatique, logiques temporelles, model-checking, systèmes d'états infinis, réseaux de Petri, automates communicants

Abstract: This thesis is about the verification problem of systems having an infinite number of states. These systems can be described by several formalisms like process algebras or automata together with unbounded data-structures (push-down automata, Petri nets or communicating finite-state machines).

In a first part of the thesis we study the characterization of classes of infinite-state systems and properties for which the verification problem is decidable. First, we consider the complexity of the verification problem of the linear-time mu-calculus for Petri nets. Then, we define temporal logics which allow to express non-regular properties containing linear constraints on the number of occurrences of events. These logics are more expressive than known logics in this domain. We show in particular that the verification problem of a logic which is more expressive than the linear-time mu-calculus is decidable for classes of systems like push-down automata and Petri nets.

A second part of the thesis is dedicated to communicating finite-state machines. Their verification problem is in general undecidable. We apply the symbolic analysis principle to these systems. We propose finite structures which allow to represent and manipulate infinite sets of configurations of these systems. These structures allow to calculate the exact effect of a repeated execution of every circuit in the transition graph of the system. Thus, every circuit of the transition graph of the system can be considered as a new "transition" of the system. We use this result to accelerate the computation of the set of reachable states of a system in order to increase the chance of termination.
Keywords: automatic verification, temporal logic, model-checking, infinite-state systems, Petri nets, communicating finite-state machines

tel-00004890, version 1 - 19 Feb 2004