Vérification d'automates étendus : algorithmes d'analyse symbolique et mise en oeuvre

Aurore Annichini Collomb

Université Grenoble i – Joseph Fourier
u.f.r. d'informatique et de mathématiques appliquées

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE I

Discipline : INFORMATIQUE

préparée au laboratoire **Vérimag**

présentée et soutenue publiquement

par

Aurore ANNICHINI COLLOMB

le 12 décembre 2001

TITRE

Vérification d'automates étendus : algorithmes d'analyse symbolique et mise en œuvre

Directeur de thèse :

Ahmed BOUAJJANI

JURY

Jacques Voiron,	Président
Bernard Boigelot,	Rapporteur
Andréas Podelski,	Rapporteur
Ahmed Bouajjani,	Directeur de thèse
Claude Jard,	Examinateur
Joseph Sifakis,	Examinateur

Remerciements

J'aimerai tout d'abord remercier ceux qui m'ont permis d'accéder au titre de docteur en informatique. Merci à

Monsieur Jacques Voiron, Professeur à l'Université Joseph Fourier (Grenoble 1), de m'avoir fait l'honneur de présider le jury de cette thèse.

Messieurs Bernard Boigelot, Professeur à l'Université de Liège, et Andréas Podelski, Professeur à l'Institut d'Informatique de Saarbrück, pour avoir accepter de juger ce travail et d'en être les rapporteurs.

Messieurs Claude Jard, Directeur de recherche CNRS à l'IRISA (Rennes), et Joseph Sifakis, Directeur de recherche CNRS à Verimag, de m'avoir fait l'honneur de participer à mon jury.

Monsieur Ahmed Bouajjani, Professeur à l'Université Paris 7, qui m'a conseillé tout au long de ce travail.

Cette thèse a été effectuée au laboratoire Verimag. Je remercie Monsieur Joseph Sifakis, de m'avoir accueilli dans son laboratoire et permis d'intégrer une formidable équipe. Ce travail étant bien entendu le fruit d'une équipe je tiens à remercier ici Mihaela Sighireanu qui a beaucoup œuvré, entre autre, pour le succès de TReX, Eugène Asarin et Yassine Lakhnech sans qui certaines parties de cette thèse n'auraient pas été si loin.

Merci aussi à tous ceux qui m'ont aidé par leur présence et leur soutien technique ou psychologique, merci à mes collègues de Verimag toujours présents (certains particulièrement, ils se reconnaitront).

Enfin, je tiens à remercier particulièrement ma famille qui m'a toujours soutenue. Merci pour la relecture ! Merci à mon mari pour son aide de tous les instants et surtout à la fin.

Table des matières

Chapitre 1

Introduction

Qui aujourd'hui n'a pas de téléphone portable ou n'est pas connecté à Internet ? Si ce n'est pas encore tout le monde, la croissance du nombre d'applications informatiques entrant dans notre vie quotidienne est exponentielle. Déjà des centrales nucléaires automatiques et bientôt les automobiles entièrement contrôlées par ordinateurs.

Pourtant comment avoir pleinement confiance en toutes ces applications informatiques ? Souvent une telle application est l'association de plusieurs composantes (chacune ayant une tâche prédéfinie) pouvant communiquer entre elles. C'est ce que nous appelons *systèmes distribués*. Il n'est pas aisé pour un humain d'appréhender l'ensemble des comportements de ce système. Pourtant ce dernier doit répondre à certaines exigences de fonctionnement appelées *spécifications*, comme par exemple être sûr que les messages envoyés par une composante vont bien être reçus par leurs destinataires. De plus, une erreur de ces systèmes, si elle est le plus souvent bénine (une machine à café peut donner du thé au lieu d'un café par exemple) peut, dans le cadre de ce que nous appelons *systèmes distribués critiques*, avoir de graves conséquences en termes de vies humaines (défaillance d'un contrôleur de centrale nucléaire par exemple). Un système critique est en effet une application informatique qui doit correspondre en tout point à la spécification (fonctionnement) attendue et surtout (d'où le nom critique) ne doit pas avoir de comportement dangereux dans les cas inattendus. La spécification d'un système peut être vue comme un *ensemble de propriétés* que doit satisfaire ce dernier pour être considéré conforme.

C'est pour vérifier qu'un système va se comporter correctement même (ou surtout) dans les cas imprévus que des méthodes dites *méthodes formelles de validation du logiciel* se sont développées ces dernières années.

Il existe deux grandes techniques de méthode formelle de validation qui sont le *test* et la *vérification*.

Le test consiste à positionner le système dans une situation de référence et observer ses comportements. En multipliant le nombre de tests dans des situations différentes, nous pouvons ainsi nous donner une certaine confiance dans le système. Pourtant, il est souvent difficile d'imaginer des situations imprévues pour ce système et donc de couvrir l'ensemble des situations possibles.

La vérification de système, quant à elle, consiste à vérifier que ce dernier est conforme à ses spécifications *quelle que soit la situation* à laquelle il est soumis. Là encore, deux approches sont possibles :

- *Approche déductive* (ou *theorem-proving*). Cette approche a été introduite par Hoare [Hoa69]. Elle est basée sur une preuve mathématique (en utilisant des axiomes et des règles d'inférence). Elle consiste à écrire le système ainsi que les propriétés à vérifier dans un modèle de sémantique axiomatisable et à démontrer que les propriétés peuvent être obtenues à partir du système en utilisant les règles d'inférence. Cette approche est utilisée dans des outils tels que Coq [CH88, HKPM97] ou PVS [ORS92]. L'avantage de cette approche réside dans sa généralité. Par contre, elle demande à l'utilisateur d'être capable de modéliser ses systèmes suivant un modèle mathématique donné pour pouvoir guider la vérification tout au long de la preuve. Cette technique n'est donc pas automatique et ne peut pas être utilisée par tous.
- *Approche algorithmique* (ou *model-checking*). Elle a été présentée notamment par [QS82, CES86]. Elle consiste à montrer que l'ensemble des comportements décrits par un système est un modèle, au sens logique du terme, pour les propriétés considérées. Le processus de vérification se décompose en fait en deux phases : la *compilation* d'un système vers un modèle mathématique, puis la *vérification* des propriétés sur ce modèle.

  Le modèle mathématique le plus souvent utilisé est le modèle des *systèmes de transitions étiquetés*. Ce modèle est un graphe dirigé dont les nœuds représentent les *états* (ou *configurations*) du système et dont les arêtes (*étiquetées*), décrivent les comportements possibles de ce système. Les propriétés sont quant à elles exprimées dans une logique temporelle (logique permettant de raisonner sur l'évolution d'un système dans le temps) telle que LTL, CTL et CTL* [Pnu77, Eme90].

Que ce soit le test ou la vérification, la première étape pour pouvoir vérifier un système est de modéliser formellement ce dernier. La question qui se pose alors est de savoir quelles hypothèses sont faites dans ce modèle et d'être convaincu que les déductions issues du système modélisé (abstrait) peuvent être corrélées au système réel (concret).

Sujet de la thèse :

Nous nous sommes principalement concentrés dans cette thèse à la vérification de protocoles de communication (téléphonie mobile, internet). Ces protocoles fonctionnent sur le principe d'envoi de messages entre deux parties par l'intermédiaire de canaux non fiables (i.e. pouvant perdre ces messages). Pour être certain que tous les messages ont bien été reçus, les techniques généralement employées consistent à réémettre les messages potentiellement perdus (le nombre de réémissions peut être fixe ou non borné), et/ou à attendre un laps de temps déterminé (au moins le temps physique de transmission entre les deux parties) avant de conclure que la transmission d'un message a échoué (souvent l'échec est déclaré lorsque l'une des deux parties ne reçoit pas d'acquittement pour le message qu'elle vient d'envoyer).

Ainsi, nous voulions travailler sur un modèle pouvant manipuler à la fois des files d'attente (modélisation des messages), des compteurs (modélisation de la réémission) et des horloges (modélisation des temps d'attente). Nous avons choisi de nous placer dans le cadre du model-cheking, c'est-à-dire trouver une méthode permettant d'analyser le plus automatiquement possible ces systèmes hétérogènes (composés de données dont le traitement n'est pas le même).

D'un autre côté, un protocole de communication est souvent paramétré par un cer-

tain nombre de variables (durée de transmission sur le canal, temps d'attente avant la réémission d'un message par exemple). Dans la plupart des techniques utilisées aujourd'hui (détaillées dans le paragraphe ci-dessous) ces paramètres doivent être instanciés pour pouvoir analyser le système. Les propriétés sont donc vérifiées pour des instances de ces paramètres, ainsi à chaque changement de valeurs de ces derniers il faut recommencer la vérification du système. Nous avons, pour notre part, cherché à trouver automatiquement les conditions sur les paramètres permettant d'affirmer que pour toutes valuations (valeurs données aux paramètres) satisfaisant ces contraintes le système vérifie ses propriétés. Pour cela, nous devions ajouter à notre représentation un moyen de manipuler des contraintes sur les paramètres.

Qu'il s'agisse des compteurs, des horloges ou des files, ces données peuvent être infinies, c'est pourquoi il était indispensable d'utiliser les *méthodes symboliques* de model-checking développées ces dernières années.

Model-checking :

Dans le cas des systèmes finis (pour lesquels les valeurs des données restent dans un ensemble fini), beaucoup de résultats ont été prouvés pour différentes logiques temporelles et des outils performants ont été créés [CE81, QS82, CES86, VW86, Mac93]. Pourtant, qui dit fini ne dit pas forcément simple à analyser. En effet, la complexité des systèmes a fortement augmenté et avec elle la taille des modèles mathématiques travaillant sur l'ensemble des états possibles, c'est ce que l'on a appelé *l'explosion combinatoire des états*. De nombreux travaux ont été effectués pour trouver des solutions à ce problème, par exemple en ne représentant qu'une partie du modèle en mémoire [JJ89, FJJM92], en réduisant le nombre de chemins à explorer [God90, GW94, Pel94, KM97, CJM00], en décomposant le système à vérifier [Pnu85, Win90], en utilisant des techniques d'abstraction [GL93, Lon93] ou encore en utilisant des représentations symboliques compactes du modèle [CBM89, BCM$^+$90, Bry92, Mac93].

Grâce à toutes ces techniques de nombreux outils ont été réalisés et ont permis de vérifier des systèmes finis comportant jusqu'à 10^{20} états [BCM$^+$90].

Les systèmes ne sont pourtant pas à fortiori finis. En effet, ils peuvent contenir des variables à valeurs dans des domaines à priori *non bornés* telles que des compteurs, des files, des piles, des horloges, $\cdots$. Or la vérification des systèmes de transitions étiquetés infinis se heurte à un obstacle théorique : l'*indécidabilité* des problèmes liés à la vérification.

Pour contourner ce problème, deux techniques se sont développées ces dernières années.

La première consiste à restreindre l'expressivité des classes de systèmes considérés afin d'obtenir des résultats de décidabilité pour la vérification. Ainsi, les résultats suivants ont été prouvés pour quelques classes particulières de systèmes :

- les *systèmes à files d'attente avec perte* pour lesquels le problème de l'accessibilité est décidable [AJ93, CFP96]. L'ensemble des états atteignables peut être représenté par un ensemble régulier [ABJ98]. Pourtant bien que le problème d'accessibilité ait été prouvé décidable, Mayr a montré [May00] que la caractérisation des contenus de files par ensemble régulier n'est pas effectivement constructible. Abdulla, Boasson et Bouajjani [ABB01] ont caractérisé les classes de systèmes à files d'attente avec perte dont l'ensemble des états atteignables est effectivement reconnaissable par un

ensemble régulier.
- les *automates temporisés* [AD94] pour lesquels le problème du test à vide est décidable. Les logiques telles que LTL, CTL* sont aussi décidables [Eme90, AH92].

La seconde consiste à trouver des méthodes (dans ce cas incomplètes) permettant de réduire les cas d'indécidabilité, c'est-à-dire permettant d'obtenir dans certains cas particuliers la vérification de la propriété sur le système infini. Deux principales méthodes se sont développées :
- L'*abstraction* ou l'*interprétation abstraite* [CC77, CH78, BBF+00] est une méthode basée sur le fait de calculer une approximation supérieure de l'ensemble des états atteignables. Cette approximation peut permettre de vérifier une propriété [Hal93].
- L'*accélération* est basée sur la notion de représentation symbolique des ensembles de valeurs des variables du système. Ce qui permet dans certains cas de faire terminer le calcul des états atteignables. Cette méthode consiste en effet à calculer en un pas l'ensemble (potentiellement infini) de valeurs que peuvent prendre les variables par l'application (infinie) d'un certain nombre de transitions du système. C'est une notion qui a été développée au cours de ces dernières années pour les systèmes à compteurs [BW94], pour les systèmes à files d'attente [BGWW97, BH97, AAB99b]

Ainsi l'analyse d'un système peut se faire en combinant l'abstraction et les méthodes d'accélération [AAB+99a].

De plus, bien que l'on soit dans une classe particulière (systèmes à files d'attente avec perte par exemple) il n'est pas exclu d'utiliser des techniques d'accélération ou d'abstraction permettant de pouvoir calculer plus souvent (notamment pour les systèmes dont l'ensemble des états atteignables est très gros) l'ensemble des états atteignables.

Contributions de la thèse :

Nous nous sommes placés dans le cadre du model-checking de systèmes infinis. Les principaux résultats théoriques sont :
- une extension des matrices de bornes (DBM) [Dil89, Yov98] — utilisées dans le cadre de la représentation de valeurs pour les horloges — pour l'analyse de systèmes à compteurs et à horloges contenant des paramètres. Nous avons pour cela redéfini la notion de matrice et les opérations qui y sont rattachées.
- une méthode générale et automatique d'analyse de systèmes complexes (systèmes faisant intervenir différentes sortes de données telles que compteurs, horloges ou files d'attente). Cette analyse commence par une phase d'abstraction du modèle permettant d'obtenir un système plus simple. La seconde phase consiste en une analyse symbolique de ce système abstrait. Dans le cadre des systèmes à compteurs et à horloges paramétrés, l'analyse symbolique repose sur un algorithme d'accélération basé sur la détection de périodicité. Cette périodicité est détectée en observant les valeurs des compteurs (ou des horloges) successives. Cette méthode (exacte) permet dans de nombreux cas d'obtenir l'ensemble des états atteignables.

Sur le plan pratique, nous avons implanté un prototype TReX mettant en œuvre ces méthodes. Pour cela, nous avons créé un outil dans un souci de généricité permettant ainsi l'utilisation facile d'autres structures de données ou de méthodes d'accélération. Ainsi

nous pouvons analyser des systèmes complexes faisant intervenir plusieurs données ayant des domaines de définitions différents et potentiellement infinis. Nous avons réalisé une librairie pour les expressions régulières représentant les contenus des files d'attente avec perte et une librairie pour les matrices de bornes paramétrées. Cet outil nous a permis de vérifier des protocoles non triviaux tels que le protocole de retransmission bornée ou le protocole de routage ("Root Contention IEEE 1394 [CAS01]).

Plan de la thèse :

Le chapitre 2 présente une définition générale des systèmes manipulés que nous avons appelé *automates étendus*. Ce sont des automates d'états finis manipulant des variables. Ces variables seront pour notre part des compteurs, des horloges ou des files d'attente. Les compteurs et les horloges peuvent, dans notre modèle, être comparés à des paramètres (variables non instanciées).

Pour analyser des systèmes complexes (au sens où ils manipulent de nombreuses données différentes) une première idée est de se ramener à des cas de systèmes plus simples. En fait, se ramener à des systèmes que nous savons analyser. C'est l'idée de l'abstraction de modèle que nous développons dans le chapitre 3.

Ayant obtenu un modèle plus simple par abstraction, nous traitons ensuite du problème de la vérification de propriétés de sûreté sur les automates étendus, ainsi que du problème de la synthèse de contraintes sur les paramètres. Nous voyons qu'il est possible de définir des principes généraux quant à la représentation des valeurs pour les variables (représentation symbolique satisfaisant de *bonnes* propriétés), la façon de calculer l'ensemble des états atteignables par application des transitions du système et la méthode d'accélération.

Nous développons dans le chapitre 4, la représentation symbolique choisie dans le cadre des systèmes à compteurs et à horloges paramétrés (les compteurs et les horloges peuvent être comparés ou assignés à des expressions contenant des paramètres). Cette représentation est une extension des matrices de bornes utilisées pour l'analyse des systèmes temporisés sans paramètres. Nous explicitons la méthode d'accélération que nous avons choisi pour cette représentation.

Le chapitre 5 s'intéresse au problème plus particulier de la vérification de propriété de vivacité pour les systèmes à compteurs et à horloges paramétrés. Nous voyons dans un premier temps que nous pouvons utiliser les méthodes décrites pour le calcul des états atteignables pour pouvoir répondre au problème de la vérification de propriétés de vivacité. Pour cela, il nous faudra pouvoir décider de l'itérabilité des boucles du graphe représentant le comportement du système analysé (graphe qui est contruit lors du calcul des états atteignables d'un système).

Le chapitre 6 expose la structure de données utilisée dans le cadre des systèmes à files d'attente avec perte. Cette représentation est un sous-ensemble des expressions régulières. Nous donnons l'algorithme d'accélération développé pour cette représentation.

Le chapitre 7 offre une vue de TREX notre prototype implantant les méthodes décrites dans les chapitres précédents. Nous commençons par décrire de façon générale l'outil puis nous entrons dans quelques détails d'implantation, pour finir par donner quelques résultats et performances. Nous finissons ce chapitre par une comparaison de notre outil avec d'autres outils de vérifications existants.

Finalement, nous concluons cette thèse dans le chapitre 8.

Chapitre 2

Automates étendus

Dans ce chapitre nous introduisons le modèle général des *automates étendus*. C'est au travers de ce modèle que nous allons analyser et vérifier les services de systèmes distribués tels que les protocoles de communication.

Un automate étendu est un automate d'états fini sur lequel on ajoute la possibilité de manipuler des données extérieures telles que compteurs, horloges ou files d'attente.

Nous donnons ici une définition générale de la notion d'automate étendu. Il est possible d'instancier cette définition pour manipuler entre autres des compteurs, des horloges ou des files d'attente.

La première partie de ce chapitre présente la notion de système de transitions étiqueté (utile pour la compréhension de la sémantique des automates étendus) et la notion d'arithmétique. Nous présentons ensuite la syntaxe et la sémantique du modèle des automates étendus et nous voyons finalement quelques exemples de tels automates.

2.1 Préliminaires

2.1.1 Système de transitions étiqueté

Nous définissons dans cette partie la notion de système de transitions étiqueté. Ce sont des graphes dirigés étiquetés. Un tel système est composé d'un ensemble d'états de contrôle (nœuds du graphe) et d'un ensemble de transitions reliant ces états. Les transitions sont étiquetés sur un vocabulaire donné. On pourra consulter [Arn92] pour une description plus détaillée des systèmes de transitions étiquetés. Plus formellement, nous définissons les systèmes de transitions étiquetés de la manière suivante.

Définition 1 (Système de transitions étiqueté — STE)

Un système de transitions étiqueté est un triplet $(Q, L, \longrightarrow)$ où :

- *Q est un ensemble d'états de contrôle.*
- *L est un ensemble fini d'étiquettes.*
- *$\longrightarrow \subseteq Q \times L \times Q$, est un ensemble de transitions.*

Pour des raisons que nous voyons dans les chapitres suivants (ce sont notamment des raisons de manipulation et de construction), nous définissons ci-après la caractérisation d'un système de transitions dit *fini*.

Définition 2 (Système de transitions étiqueté fini)
Un système de transitions étiqueté est fini si Q est un ensemble fini.

Vérifier des propriétés sur le modèle des automates étendus c'est observer les exécutions de ce modèle. Une exécution ou plus précisément une *séquence d'exécution* pour un système de transitions étiqueté est une séquence de transitions de ce système. Quelquefois nous nommons cette séquence *chemin*.

Définition 3 (Séquence d'exécution σ)
Soit $S = (Q, L, \longrightarrow)$ un système de transitions étiqueté. Une séquence d'exécution σ est de la forme suivante :
- $q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} q_2 \cdots$. $\forall\, i \geq 0\ q_i \xrightarrow{l_i} q_{i+1}$ (séquence infinie)
ou de la forme
- $q_0 \xrightarrow{l_0} q_1 \cdots q_n \xrightarrow{l_n} q_{n+1}$ et $\forall\ q' \in Q,\ \forall\ l \in L,\ \nexists\ q_{n+1} \xrightarrow{l} q'$ (séquence finie maximale).

On note par $Exec(S)$ l'ensemble de toutes les séquences d'exécution du système S.

Pour chaque séquence d'exécution, nous lui associons une séquence d'étiquettes (ce sont celles, dans l'ordre de la séquence d'exécution, qui apparaîssent dans cette dernière). Une telle séquence d'étiquettes est appelée *trace*.

Définition 4 (Trace d'une séquence d'exécution)
Soit S un système de transitions étiqueté. Soit σ une séquence d'exécution de S.
$trace(\sigma) \in L^*$ est la projection sur les étiquettes de la séquence d'exécution σ.
- $trace(\sigma) = l_1 l_2 \cdots$ si σ est une séquence infinie.
- $trace(\sigma) = l_1 l_2 \cdots l_m$ si σ est une séquence finie maximale.

Nous pouvons donc maintenant définir l'ensemble des traces d'un système de transitions étiqueté.

Définition 5 (Traces d'un STE)
Soit S un système de transitions étiqueté.
Alors $Traces(S) = \{trace(\sigma) \mid \sigma \in Exec(S)\}$.

Définition 6 *Soit S un système de transitions étiqueté.*

- *S est déterministe si et seulement si : $\forall\ q \in Q,\ \forall l \in L,\ \exists! q' \in Q.\ q \xrightarrow{l} q'$*
- *S est complet si et seulement si : $\forall q \in Q\ \forall l \in L\ \exists q' \in Q . q \xrightarrow{l} q'$*

2.1.2 Arithmétique

Soit X un ensemble de variables (interprétées sur les réels) et x une variable de cet ensemble. L'ensemble des *termes arithmétiques* sur X, dénoté $AT(X)$ est défini par la grammaire suivante :

$$t ::= 0 \mid 1 \mid x \mid t - t \mid t + t \mid t * t$$

On note par $ATl(X) \subset AT(X)$ l'ensemble des termes linéaires : ces termes sont construits sans l'utilisation de la multiplication $(*)$ entre deux variables. Seule la multiplication entre une constante et une variable est autorisée.

Soit $\overrightarrow{d}$ un vecteur de réels représentant la valeur de chacune des variables de X. Ce vecteur est appelé *valuation*. Ainsi, soit $\overrightarrow{d}$ une valuation des variables de X, la fonction

$eval(t, \overrightarrow{d})$ permet de calculer la valeur représentée par le terme t où les opérations $+$, $-$, $*$ sont les opérations usuelles sur les réels et $eval(x, \overrightarrow{d}) = \overrightarrow{d}(x)$ est la valeur de la variable x dans la valuation $\overrightarrow{d}$.

L'ensemble des *formules arithmétiques de premier-ordre* sur X dénoté $F(X)$ est défini par la grammaire suivante :

$$\phi ::= vrai \mid t \leq t \mid \neg\phi \mid \phi \vee \phi \mid \exists x.\phi \mid Est\_entier(t)$$

Les symboles $faux$, $\forall$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$, entre autres, se dérivent de manière usuelle à partir des symboles définis dans cette grammaire (par exemple, $\neg\exists\neg \equiv \forall$).

Soient $x \in X$ une variable, d_x une valeur possible de cette variable et $\phi \in F(X)$ une formule arithmétique, nous notons par $\phi[x/d_x]$ la formule ϕ dans laquelle nous avons substitué la variable x par sa valeur d_x. Par abus de langage, si $X_0 = \{x_1, x_2, \cdots, x_l\}$ est un sous-ensemble de X, $\overrightarrow{d} = (d_{x_1}, d_{x_2}, \cdots, d_{x_l})$ une valuation représentant leur valeur et $\phi \in F(X)$ une formule arithmétique nous écrivons alors de la même manière $\phi[X_0/\overrightarrow{d}]$ pour représenter la formule ϕ dans laquelle nous avons substitué les variables de X_0 par leur valeur.

Soit $\overrightarrow{d}$ une valuation des variables de X, la notation $\overrightarrow{d} \models \phi$ exprime que $\overrightarrow{d}$ satisfait ϕ ou que ϕ est vraie pour la valuation $\overrightarrow{d}$. Cette relation de satisfaction est définie par induction sur la structure des formules de la manière suivante :

$$\begin{array}{lcl}
\overrightarrow{d} \models vrai & & \\
\overrightarrow{d} \models t_1 \leq t_2 & \Leftrightarrow & eval(t_1, \overrightarrow{d}) \leq eval(t_2, \overrightarrow{d}) \\
\overrightarrow{d} \models \neg\phi & \Leftrightarrow & \overrightarrow{d} \not\models \phi \\
\overrightarrow{d} \models \phi_1 \vee \phi_2 & \Leftrightarrow & \overrightarrow{d} \models \phi_1 \text{ ou } \overrightarrow{d} \models \phi_2 \\
\overrightarrow{d} \models \exists x.\phi & \Leftrightarrow & \overrightarrow{d} \models \phi[x/\overrightarrow{d}(x)] \\
\overrightarrow{d} \models Est_entier(t) & \Leftrightarrow & eval(t, \overrightarrow{d}) \in \mathbb{N}
\end{array}$$

$Est\_entier(t)$ est donc un prédicat permettant de savoir si l'évaluation du terme t est une valeur entière.

Le sous-ensemble des formules de $F(X)$ dans lesquelles le prédicat Est_entier n'apparaît pas est appelé *arithmétique du premier ordre sur les réels*, dénoté $FR(X)$. Le sous-ensemble de $FR(X)$ sans l'utilisation de la multiplication est appelé *arithmétique linéaire sur les réels*, dénoté $FRL(X)$.

Il est possible de définir les mêmes fragments d'arithmétique en interprétant les variables sur les entiers.

Ainsi, le sous-ensemble des formules de $F(X)$ sans l'utilisation du prédicat Est_entier est l'ensemble des *formules arithmétiques de premier-ordre sur les entiers*, dénoté $FE(X)$.

Le fragment de $FE(X)$ des formules sans multiplication $(*)$ est appelé *arithmétique linéaire sur les entiers* (ou arithmétique de Presburger [Pre29]), dénoté $FEL(X)$.

Pour l'ensemble de ces fragments de l'arithmétique il existe des résultats de décidabilité

pour la satisfaction d'une formule que nous rappelons dans le tableau ci-après.

Arithmétique	Satisfaisabilité
F	indécidable
FR	décidable [Tar51, ACM84, Wei93]
FRL	décidable [Tar51]
FE	indécidable
FEL	décidable [Pre29]

Dans la suite nous désignons par $F(X)$ toute arithmétique définie précédemment.

Soit $\overrightarrow{\mathrm{d}}$ une valuation des variables de X et ϕ une formule de $F(X)$, $[\![\phi]\!]_{\overrightarrow{\mathrm{d}}}$ dénote alors la satisfaisabilité de ϕ pour les valeurs définies par $\overrightarrow{\mathrm{d}}$, c'est-à-dire $\overrightarrow{\mathrm{d}} \models \phi$. Par exemple, $[\![\ \exists x.x < 5\ ]\!]_{(x=3)} = vrai$. Nous définissons alors $[\![\phi]\!] = \{\overrightarrow{\mathrm{d}} \mid [\![\phi]\!]_{\overrightarrow{\mathrm{d}}}\}$ représentant l'ensemble des valuations qui satisfont la formule ϕ.

Pour toutes les arithmétiques, nous supposons qu'il existe une fonction *sat* définie de la manière suivante :

$$sat(\phi) = \begin{cases} vrai & \text{si } [\![\phi]\!] \neq \emptyset \\ faux & \text{sinon} \end{cases}$$

La fonction *sat* retourne vrai lorsqu'il existe au moins une valuation sur les variables satisfaisant la formule.

2.2 Modèle des automates étendus

Un automate étendu est un automate fini sur lequel nous ajoutons la possibilité de manipuler des données telles que compteurs, horloges ou files d'attente. Nous considérons ici des programmes ayant une *partie contrôle* et une *partie donnée*. La partie contrôle est modélisée par un graphe fini de contrôle dont les transitions sont étiquetées par les instructions du programme. La partie donnée est modélisée par des variables et des paramètres.

Le programme peut agir sur les variables grâce à des opérations prédéfinies, ces variables peuvent être testées durant le déroulement du programme. Les résultats de ces tests influencent alors ce déroulement. En effet, le modèle des automates étendus est basé sur le principe des automates à commandes gardées : pour pouvoir appliquer une transition il faut que les valeurs des variables satisfassent des conditions exprimées par ce que nous appelons *garde*. Si les variables ne satisfont pas la garde, l'automate ne change pas d'état de contrôle. Dans le modèle que nous définissons ci-dessous nous avons intégré les gardes dans les opérations.

Les paramètres sont des variables non instanciables : ils gardent leur valeur initiale tout au long de l'analyse. Les gardes peuvent contenir des comparaisons entre les variables et les paramètres, il sera aussi possible d'affecter les variables avec des paramètres.

Pour l'instant nous ne donnons pas plus d'informations sur ce que sont exactement les données et les paramètres, nous verrons dans les chapitres suivants les restrictions que nous leurs apportons (représentation, manipulation).

Plus formellement un programme est décrit par un automate étendu défini de la manière suivante. Les modèles que nous définissons ainsi sont dits *modèles concrets*, c'est la modélisation que nous faisons du monde réel.

Définition 7 (Automate Etendu)

Un automate étendu est un n-uplet $\mathcal{A}$ = (Q, $\mathcal{X}$, $\mathcal{P}$, L, Op, T) tel que :

- *Q est un ensemble fini d'états de contrôle.*
- *$\mathcal{X}$ est un ensemble fini de variables de dimension n_x. Chaque variable est interprétée sur un domaine $\mathbb{D}$ potentiellement infini. Les variables sont notées $x_1, x_2, \cdots, x_{n_x}$. On note par $\perp$ le contenu des variables indéfini.*
- *$\mathcal{P}$ est un ensemble fini de paramètres de dimension n_p. Chaque paramètre est interprété sur le domaine $\mathbb{D}$. Les paramètres sont dénotés par $p_1, p_2, \cdots, p_{n_p}$.*
- *L est un ensemble fini d'étiquettes.*
- *Op est un ensemble (possiblement infini) d'opérations sur les variables. Chaque opération $op \in Op$ est une fonction $\mathbb{D}^{n_x} \longrightarrow \mathbb{D}^{n_x} \cup \perp$.*
- *$T \subseteq Q \times L \times Op \times Q$ est l'ensemble des transitions du système.*

Soit (q_1, l, op, q_2) une transition de l'automate, q_1 est *l'état source* de la transition et q_2 en est *l'état destination.*

2.3 Sémantique

Nous définissons dans cette partie la sémantique opérationnelle des automates étendus. A chaque automate étendu $\mathcal{A}$ nous allons associer un système de transitions étiqueté $\mathcal{S}$ qui représente l'ensemble des séquences d'exécution de $\mathcal{A}$.

La sémantique manipule les variables et les paramètres en considérant leurs valeurs à un moment donné dans l'exécution du système. Nous appelons *valuation* des variables $\mathcal{X}$ le vecteur $\overrightarrow{\mathrm{d}} = (d_{x_1},\ d_{x_2},\ \cdots,\ d_{x_{n_x}})$ tel que $\forall i \in [1,\ n_x]\ d_{x_i} \in \mathbb{D}$. La valeur $\overrightarrow{\mathrm{d}}(x_i)$ dénote la valeur de la variable x_i dans la valuation $\overrightarrow{\mathrm{d}}$. La notion de valuation $\overrightarrow{\mathrm{p}}$ sur les paramètres est définie de la même manière.

Une *configuration* $\gamma = (q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$ est l'association d'un état de contrôle de l'automate étendu $q \in Q$, d'une valuation des variables $\overrightarrow{\mathrm{d}} \in \mathbb{D}^{n_x}$ et d'une valuation des paramètres $\overrightarrow{\mathrm{p}} \in \mathbb{D}^{n_p}$.

A chaque pas, les changements de configurations possibles sont déterminés par l'application des transitions sortant de l'état de contrôle de la configuration courante. Formellement la sémantique d'un automate étendu est un système de transitions dont les nœuds sont les configurations et dont les transitions correspondent à l'application d'une transition de l'automate.

Définition 8 (Sémantique d'un automate étendu)

La sémantique d'un automate étendu $\mathcal{A}$ =(Q, $\mathcal{X}$, $\mathcal{P}$, L, Op, T) est le système de transitions étiqueté $\mathcal{S}(\mathcal{A}) = (C,\ L,\ R)$, appelé graphe de l'espace des configurations atteignables *tel que :*

- *$C = Q \times \mathbb{D}^{n_x} \times \mathbb{D}^{n_p}$ est l'ensemble des configurations possibles de $\mathcal{A}$. Chaque état $\gamma = (q, \overrightarrow{d}, \overrightarrow{p}) \in C$ est composé d'un état de contrôle $q \in Q$, d'une valuation des variables $\overrightarrow{d} \in \mathbb{D}^{n_x}$ et d'une valuation des paramètres $\overrightarrow{p} \in \mathbb{D}^{n_p}$. Puisque le domaine $\mathbb{D}$ peut être infini, il est alors possible que l'ensemble C soit infini.*

– $R \subseteq C \times L \times C$ *est la* relation d'atteignabilité en un pas *de* S.

Un triplet $((q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}),\ l,\ (q', \overrightarrow{\mathrm{d}'}, \overrightarrow{\mathrm{p}'}))$ appartient à R, ce qui s'écrit $(q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}) \xrightarrow{l} (q', \overrightarrow{\mathrm{d}'}, \overrightarrow{\mathrm{p}'})$, s'il existe une transition $(q, l, op, q') \in T$ telle que

$$\overrightarrow{\mathrm{d}'} = op(\overrightarrow{\mathrm{d}}) \ \wedge\ \overrightarrow{\mathrm{p}'} = \overrightarrow{\mathrm{p}} \ \wedge\ \overrightarrow{\mathrm{d}'} \neq \perp$$

En effet, pour toute opération et toute valuation des variables, si l'application de l'opération sur ces variables est la valuation indéfinie $\perp$ la transition $((q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}), l, (q', \perp, \overrightarrow{\mathrm{p}}))$ n'appartient pas au graphe. Nous ne représentons pas les transitions qui n'ont pas pu s'exécuter.

Soit R^* (notée $\xrightarrow{*}$) la fermeture transitive et réflexive de R.

Pour calculer l'ensemble des configurations atteignables, il est d'usage de définir une configuration γ_0 dite *configuration initiale* qui explicite l'état de contrôle dans lequel on se place pour calculer l'ensemble des configurations atteignables ainsi que les valeurs initiales des variables et des paramètres.

Une configuration γ' est dite *atteignable depuis la configuration* γ si et seulement si $\gamma \xrightarrow{*} \gamma'$.

L'*espace des configurations atteignables* (ou *espace des états*) (C, R) de $\mathcal{A}$ est le graphe (possiblement infini) dont les noeuds sont les configurations atteignables de $\mathcal{A}$ et dont les transitions correspondent à la relation d'atteignabilité en un pas R entre les états.

Nous introduisons quelques notations résumant l'atteignabilité :

Définition 9 ($Post$, Pre, $Post^*$, Pre^*)

Soient $\mathcal{A}$ *un automate étendu et* $\mathcal{S}(\mathcal{A})$ *sa sémantique. Soient* $t \in T$ *une transition de* $\mathcal{A}$ *et* $\gamma = (q, \overrightarrow{d}, \overrightarrow{p})$ *une de ses configurations, nous définissons alors :*

$$\begin{aligned}
Post_t(\gamma) &= \{\gamma' \mid \gamma \xrightarrow{l} \gamma'\} \\
Pre_t(\gamma) &= \{\gamma' \mid \gamma' \xrightarrow{l} \gamma\} \\
Post(\gamma) &= \{\gamma' \mid \exists (q, l, op, q') \in T.\ \gamma \xrightarrow{l} \gamma'\} \\
Pre(\gamma) &= \{\gamma' \mid \exists (q', l, op, q) \in T.\ \gamma' \xrightarrow{l} \gamma\} \\
Post^*(\gamma) &= \{\gamma' \mid \gamma \xrightarrow{*} \gamma'\} \\
Pre^*(\gamma) &= \{\gamma' \mid \gamma' \xrightarrow{*} \gamma\}
\end{aligned}$$

$Post^*(\gamma)$ représente l'ensemble des configurations atteignables à partir de la configuration γ et $Pre^*(\gamma)$ représente l'ensemble des configurations à partir desquelles γ est atteignable.

Remarque :

De manière générale, il nous est conceptuellement plus simple de nous représenter le comportement d'un système complexe par plusieurs sous-composantes.

Par exemple, si nous devons modéliser le fonctionnement d'un service internet permettant à une machine A d'envoyer des messages à une machine B, il est plus simple de raisonner séparément sur le comportement des deux machines. En effet, l'une doit émettre les messages, la seconde les recevoir, ce n'est pas le même principe. Pourtant, il nous faut raisonner sur le comportement global du service et non pas sur chacune de ces composantes. Nous introduisons donc ici la possibilité de décrire séparément les composantes d'un

système complexe et nous donnons alors la définition de l'automate étendu correspondant au système complet. Nous supposons ici que les composantes peuvent s'exécuter seules, nous nous plaçons dans une sémantique asynchrone. Pour communiquer entre elles, nous supposons qu'il existe un ensemble de variables $\mathcal{X}^c$ et un ensemble de paramètres $\mathcal{P}^c$ communs à toutes les composantes. Ainsi chaque composante $\mathcal{A}_i = (Q^i,\ \mathcal{X}^i,\ \mathcal{P}^i,\ L^i,\ Op^i,\ T^i)$ est un automate étendu où $\mathcal{X}^i = \mathcal{X}^i_1 \cup \mathcal{X}^c$, $\mathcal{P}^i = \mathcal{P}^i_1 \cup \mathcal{P}^c$.

De plus, lors de la modélisation d'un système, il est parfois utile de pouvoir raisonner sur une séquence de transitions en supposant que cette dernière est atomique. Cela permet en particulier de factoriser des états de contrôle. C'est pourquoi nous introduisons dans notre modèle la possibilité de déclarer des états dits *instables*, c'est-à-dire lors de l'analyse du système si un processus est dans un état instable seules les transitions de ce processus sont alors applicables, les autres processus sont alors bloqués. Ainsi, pour chaque composante, nous définissons un sous-ensemble des états de contrôle $Q_{inst} \subseteq Q^i$ qui caractérise l'ensemble des états instables.

Nous définissons ci-dessous le produit asynchrone entre deux processus pour simplifier l'écriture, la généralisation de cette opération se faisant de manière usuelle.

Définition 10 (Produit asynchrone d'automates étendus)
Soient $\mathcal{A}^1, \mathcal{A}^2$ 2 automates étendus représentant les composantes.
On peut alors définir l'automate $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2 = (Q,\ \mathcal{X},\ \mathcal{P},\ L,\ Op,\ T)$ tel que :
- *$Q = Q^1 \times Q^2$ est l'ensemble fini du produit cartésien des états de contrôle de chacun des automates $\mathcal{A}^i$ ($i \in [1,\ n]$).*
- *$\mathcal{X} = \mathcal{X}^1_1 \cup \mathcal{X}^2_1 \cup \mathcal{X}^c$ est l'union des ensembles de variables de chacun des automates. Soit n_x le nombre de variables de $\mathcal{A}$.*
- *$\mathcal{P} = \mathcal{P}^1_1 \cup \mathcal{P}^2_1 \cup \mathcal{P}^c$ est l'union des ensembles de paramètres de chacun des automates. Soit n_p le nombre de paramètres de $\mathcal{A}$.*
- *$L = L^1 \cup L^2$ est l'union des ensembles des étiquettes de chacun des automates.*
- *$Op = Op^1 \cup Op^2$ est l'ensemble des opérations de $\mathcal{A}$. Pour chaque opération $op \in Op^i$, ($i \in [1, 2]$), nous étendons les ensembles de départ et d'arrivée à l'ensemble de toutes les variables de l'automate du produit asynchrone. Cette nouvelle fonction ne modifie que les variables qui appartenaient à l'automate A^i et laisse les autres inchangées.*
- *$T \subseteq Q \times L \times Op \times Q$ tel que $((q^1,\ q^2),\ l,\ op,\ (q'^1,\ q'^2)) \in T$ si et seulement si $\forall i \in [1, 2],\ q^i, q'^i \in Q^i \wedge (\exists j \in [1, n].\exists t \in T^j.t = (q^j, l, op, q'^j) \wedge \forall l \in [1, 2].l \neq j \Rightarrow q^l = q'^l)$.*

A partir de maintenant, nous ne travaillons que sur *un* automate (éventuellement après avoir effectué le produit asynchrone).

2.4 Exemples d'automates étendus

2.4.1 Les systèmes à compteurs paramétrés

2.4.1.1 Définition

Dans le cadre des systèmes à compteurs paramétrés les variables, appelées compteurs, et les paramètres sont interprétés sur les entiers naturels ($\mathbb{D} = \mathbb{N}$). Les opérations se décomposent en deux parties. Une partie dite de test et une d'affectation. Ainsi toute opération $op \in Op$ est de la forme (o_g, o_v) où :

- $o_g \in FEL(\mathcal{X} \cup \mathcal{P})$ est une conjonction de contrainte sur les compteurs et les paramètres de la forme $c_i \# t$ ou $c_i - c_j \# t$ telles que $c_i, c_j \in \mathcal{X}$, $\# \in \{<, \leq, =, \geq, >\}$ et t est un terme arithmétique sur les compteurs et les paramètres.
- o_v est une opération d'affectation sur les compteurs de la forme $c_i := c_j + t_i$ ou $c_i := t_i$ telle que $c_i, c_j \in \mathcal{X}$ et t_i est un terme arithmétique sur les compteurs et les paramètres.

Lors d'une transition, il ne peut y avoir au plus qu'une affectation pour chacun des compteurs. On note l'affectation du compteur c_i par $o_v(c_i) = (c_j, t_i)$ où $c_j = \perp$ si l'affectation est de la forme $c_i := t_i$.

Supposons que le système contienne n_c compteurs et n_p paramètres. La sémantique d'un système à compteurs paramétrés est représentée par l'application d'une transition sur les configurations.

Nous rappelons qu'une configuration $\gamma = (q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$ est l'association d'un état de contrôle $q \in Q$, d'une valuation des compteurs $\overrightarrow{\mathrm{d}} \in \mathbb{N}^{n_c}$ et d'une valuation des paramètres $\overrightarrow{\mathrm{p}} \in \mathbb{N}^{n_p}$.

Pour qu'une transition (q, l, o_g, o_v, q') soit appliquable à partir d'une configuration $\gamma = (q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$, il faut tout d'abord que les valeurs des compteurs et des paramètres satisfassent la garde o_g. Ensuite, nous pouvons appliquer les affectations représentées par la fonction o_v.

Définition 11 (Relation de transition)
Pour chaque transition $(q_1, l, o_g, o_v, q_2) \in T$, nous définissons une relation de transition sur les configurations telle que $(q_1, \overrightarrow{d}_1, \overrightarrow{p}_1) \xrightarrow{l} (q_2, \overrightarrow{d}_2, \overrightarrow{p}_2)$ si et seulement si

$$sat(o_g[\mathcal{X}/\overrightarrow{d}_1, \mathcal{P}/\overrightarrow{p}_1]) \ \wedge \ \overrightarrow{p}_2 = \overrightarrow{p}_1 \ \wedge \ \forall c_i \in \mathcal{X}.o_v(c_i) = (c_j, b_i) \wedge$$
$$((c_j = \perp \Rightarrow \overrightarrow{d}_2(c_i) = b_i) \ \vee \ (c_j \neq \perp \Rightarrow \overrightarrow{d}_2(c_i) = \overrightarrow{d}_1(c_j) + b_i))$$

2.4.1.2 Exemple : exclusion mutuelle — l'algorithme du "Bakery"

En général, les algorithmes d'exlusion mutuelle travaillent sur un réseau de processus. Chaque processus essaie d'obtenir à tour de rôle une ressource commune. La ressource peut être un fichier dans le cadre de la gestion de fichiers ou encore une machine particulière dans le cadre de l'ordonnancement des tâches dans une usine. Il existe de nombreux algorithmes permettant d'obtenir une exclusion mutuelle. Voici une description de l'algorithme de "Bakery" [And91].

Avant de détailler l'algorithme du "Bakery", nous rappelons qu'un bon algorithme d'exclusion mutuelle est un algorithme qui satisfait les propriétés suivantes :

- *A aucun moment, il n'est possible d'avoir plus d'un processus utilisant la ressource.* C'est une propriété de sûreté exprimant l'exclusion mutuelle.
- *Si un processus demande la ressource, alors il l'aura à un moment donné plus tard dans l'exécution.* C'est une propriété de vivacité exprimant la non-famine (*starvation-freedom*).

Regardons maintenant plus en détail l'algorithme du "Bakery". Nous supposons ici que le réseau est composé de 2 processus.

Lorsqu'un processus veut entrer dans la section critique (état dans lequel il est lorsqu'il a obtenu la ressource), il doit tout d'abord prendre un ticket. Le numéro de ce dernier doit être supérieur aux numéros de tous les tickets que détiennent les autres processus (ici on peut dire l'autre processus) qui se trouvent dans la section critique ou en attente de cette dernière. Un processus prend un ticket en ajoutant 1 à la plus grande valeur de tickets existants. Ensuite, pour entrer effectivement dans la section critique il doit avoir le numéro de ticket le plus petit.

Les tickets de chaque processus pour l'accession à la ressource sont modélisés par 2 compteurs (variables partagées par les deux processus) c_1 et c_2. Puisqu'ici, nous ne nous intéressons qu'à deux processus, le processus 1 prend un ticket de numéro $c_2 + 1$ et le processus 2 le ticket de numéro $c_1 + 1$.

L'algorithme du "Bakery" est modélisé par l'automate $Bakery = Proc^1 \times Proc^2$ qui est le produit asynchrone de systèmes à compteurs représentant respectivement le comportement du processus 1 et du processus 2.

Le système $Proc^i = (Q^i, \mathcal{X}^i, \mathcal{P}^i, L^i, Op_g^i, Op_v^i, T^i)$ représentant le processus i ($i \in \{1, 2\}$) est tel que :

- $Q^i = \{repos^i,\ attente^i,\ sectionc^i\}$, l'état initial q_0^i est $repos^i$.
- Les variables utilisées par le processus i sont les compteurs partagés c_1 et c_2. La valuation initiale pour ces compteurs est $\overrightarrow{\mathrm{d}}_0 = (0, 0)$.
- $\mathcal{P} = \emptyset$, le modèle ne contient pas de paramètres.
- $L^i = \{demander^i,\ entrer^i,\ sortir^i\}$ est l'ensemble des étiquettes du système.

La figure 2.1 représente le processus 1, pour obtenir la modélisation du processus 2 il faut intervertir l'indice 1 par l'indice 2 et vice-versa. Dans cette figure, les transitions sont de la forme *label* : *garde/operation*, si l'une de ces trois catégories est vide elle n'apparaîtra pas sur la transition. De même si la valeur de la variable reste inchangée ($c_i := c_i$) nous ne le notons pas sur les transitions.

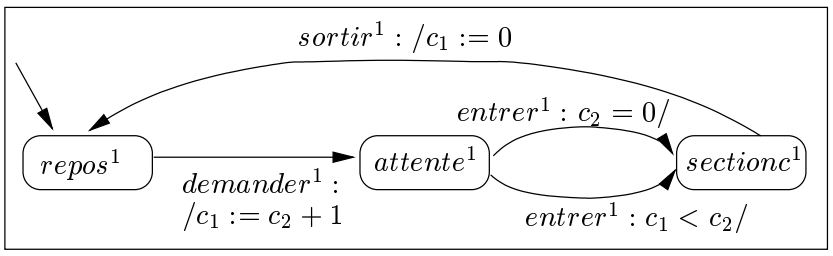


FIG. 2.1 – Modélisation de $Proc^1$

Essayons maintenant de calculer l'ensemble des configurations atteignables de ce modèle. La configuration initiale est $\gamma_0 = ((repos^1, repos^2), 0, 0)$ (que nous écrivons $((r^1, r^2), 0, 0)$ pour plus de lisibilité dans la figure 2.2). A partir de γ_0, deux transitions sont appliquables :

- La transition étiquetée $demander^1$: on obtient alors la configuration $\gamma_{01} = ((attente^1, repos^2), 1, 0)$. Le processus 1 demande la ressource.
- La transition étiquetée $demander^2$: on obtient alors la configuration $\gamma_{02} = ((repos^1, attente^2), 0, 1)$. Le processus 2 demande la ressource.

Ensuite, le processus 1 peut entrer dans la section critique à partir de γ_{01} : nous obtenons alors la configuration $\gamma_{011} = ((sectionc^1, repos^2), 1, 0)$. Sinon, à partir de γ_{02} le processus 1 peut demander la ressource : nous obtenons alors la configuration $\gamma_{021} = ((attente^1, attente^2), 2, 1)$.

Une partie de l'ensemble des configurations atteignables est représentée par la figure 2.2.

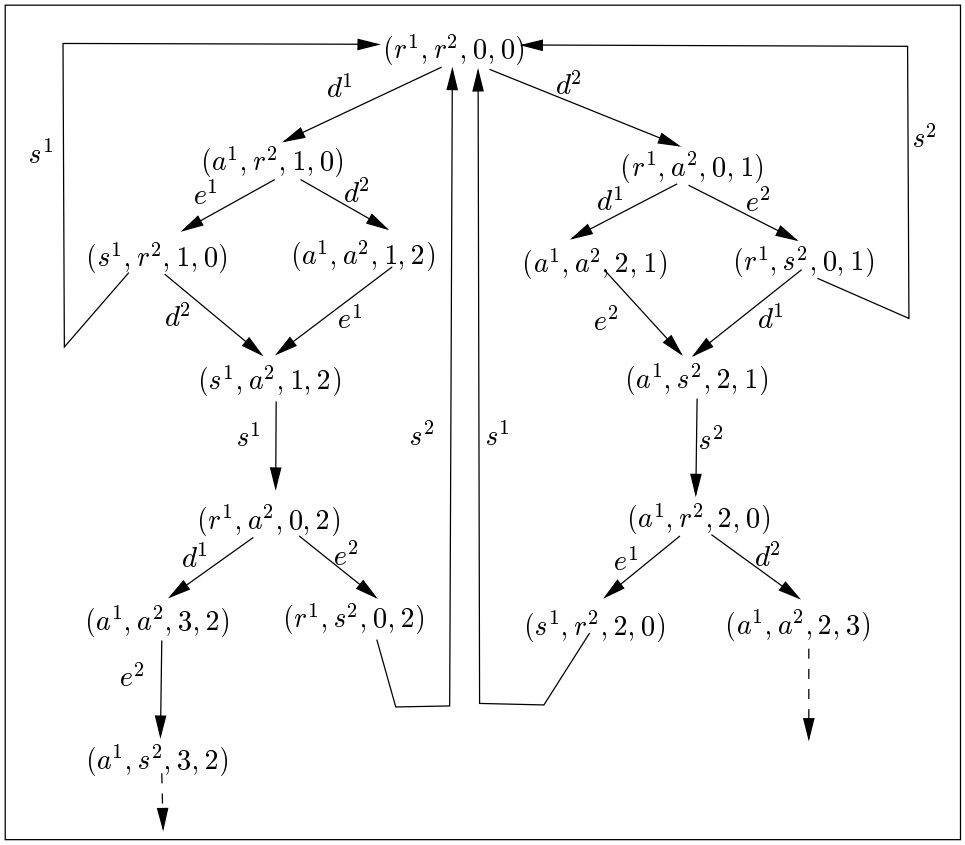


FIG. 2.2 – Une partie de l'ensemble des configurations atteignables pour le "Bakery"

L'ensemble des configurations atteignables pour cet exemple est infini, les compteurs c_1 et c_2 peuvent en effet prendre une infinité de valeurs.

2.4.2 Les systèmes à horloges paramétrées

2.4.2.1 Définition

Dans le cadre des systèmes à horloges paramétrées les variables, appelées horloges, et les paramètres sont interprétés sur les réels ($\mathbb{D} = \mathbb{R}$). Nous ajoutons à la définition d'un automate la notion d'invariant. Ainsi, $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, I, L, Op, T)$ est un système à horloges paramétrées où $\mathcal{X}$ sont des variables dont le domaine de définition est $\mathbb{R}$ et I est une fonction qui associe à chaque état de contrôle une contrainte que doivent satisfaire les horloges pour pouvoir rester dans cet état ($I : Q \longrightarrow FRL(\mathcal{X} \cup \mathcal{P})$). Les opérations se décomposent en deux parties. Une partie dite de test et une d'affectation. Ainsi toute opération $op \in Op$ est de le forme (o_g, o_v) où :

- $o_g \in FRL(\mathcal{X} \cup \mathcal{P})$ est une conjonction de contrainte sur les compteurs et les paramètres de la forme $h_i \# t$ ou $h_i - h_j \# t$ telles que $h_i, h_j \in \mathcal{X}$, $\# \in \{<, \leq, =, \geq, >\}$ et t est un terme arithmétique sur les horloges et les paramètres.
- o_v est une opération d'affectation sur les horloges de la forme $h_i := h_j$ ou $h_i := 0$ telle que $h_i, h_j \in \mathcal{X}$.

Lors d'une transition, il ne peut y avoir au plus qu'une affectation pour chacune des horloges.

Supposons que le système contienne n_h horloges et n_p paramètres. La sémantique d'un système à horloges paramétrées est représentée par l'application d'une transition sur les configurations.

Nous rappelons qu'une configuration $\gamma = (q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$ est l'association d'un état de contrôle du système $q \in Q$, d'une valuation sur les horloges $\overrightarrow{\mathrm{d}} \in \mathbb{R}^{n_h}$ et d'une valuation sur les paramètres $\overrightarrow{\mathrm{p}} \in \mathbb{R}^{n_p}$.

Pour qu'une transition (q, l, o_g, o_v, q') soit applicable à partir d'une configuration $\gamma = (q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$, il faut tout d'abord que les valeurs des horloges et des paramètres satisfassent la garde o_g. Ensuite, nous pouvons appliquer les affectations représentées par l'opération o_v et laisser passer le temps. Enfin, la valuation des horloges obtenues doit satisfaire l'invariant donné pour l'état q'. La relation de transition se décompose donc en deux parties :

- La relation de transition discrète permet d'appliquer les affectations représentées par o_v.
- La relation de transition temporelle permet de laisser passer le temps. La notion de passage de temps s'exprime en ajoutant une valeur t identique à toutes les horloges. Dans ce modèle nous supposons que les horloges avancent à la même vitesse, cette vitesse étant constante.

Définition 12 (Relation de transition discrète)

Pour chaque transition $(q_1, l, o_g, o_v, q_2) \in T$ nous définissons une relation de transition discrète sur les configurations telle que $(q_1, \overrightarrow{d}_1, \overrightarrow{p}_1) \xrightarrow{l} (q_2, \overrightarrow{d}_2, \overrightarrow{p}_2)$ si et seulement si :

$$(sat(o_g[\mathcal{X}/\overrightarrow{d}_1,\ \mathcal{P}/\overrightarrow{p}_1])\ \wedge\ (\overrightarrow{p}_2 = \overrightarrow{p}_1) \wedge$$
$$(\forall h_i \in \mathcal{X}.\ (o_v(h_i) = 0 \Rightarrow \overrightarrow{d}_2(h_i) = 0)\ \vee\ (o_v(h_i) = h_j \Rightarrow \overrightarrow{d}_2(h_i) = \overrightarrow{d}_1(h_j))) \wedge$$
$$(sat(I(q_2)[\mathcal{X}/\overrightarrow{d}_2,\ \mathcal{P}/\overrightarrow{p}_2])$$

Soient $t \in \mathbb{R}$ un temps donné et $\overrightarrow{\mathrm{d}}$ une valuation des horloges, nous notons par $\overrightarrow{\mathrm{d}} + t$ la valuation telle que les valeurs de toutes les horloges ont été augmentées de t.

Définition 13 (Relation de transition temporelle)

Nous définissons une relation de transition temporelle sur les configurations telle que $(q_1, \overrightarrow{d}_1, \overrightarrow{p}_1) \xrightarrow{\delta} (q_2, \overrightarrow{d}_2, \overrightarrow{p}_2)$ *si et seulement si :*

$$(\overrightarrow{d}_2 = \overrightarrow{d}_1 + \delta) \ \wedge \ (\overrightarrow{p}_2 = \overrightarrow{p}_1) \wedge$$
$$\forall t \in I\!R.0 \leq t \leq \delta.\ sat(I(q_2)[\mathcal{X}/(\overrightarrow{d}_1 + t),\ \mathcal{P}/\overrightarrow{p}_2])$$

La relation de transition globale est alors l'application de la transition discrète puis l'application de la transition temporelle. Ainsi, nous dirons que $(q_1, \overrightarrow{\mathrm{d}}_1, \overrightarrow{\mathrm{p}}_1) \stackrel{l}{\Longrightarrow} (q_2, \overrightarrow{\mathrm{d}}_2, \overrightarrow{\mathrm{p}}_2)$ si et seulement si il existe une transition $(q_1, l, o_g, o_v, q_2) \in T$ et un temps δ tels que $(q_1, \overrightarrow{\mathrm{d}}_1, \overrightarrow{\mathrm{p}}_1) \xrightarrow{l} (q_2, \overrightarrow{\mathrm{d}}', \overrightarrow{\mathrm{p}}') \xrightarrow{\delta} (q_2, \overrightarrow{\mathrm{d}}_2, \overrightarrow{\mathrm{p}}_2)$

Remarque : Dans la littérature et dans les outils de model-checking, plusieurs interprétations des invariants sont données. La question principale qui se pose est : est-ce que $(q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$ est une configuration si $I(q)[\mathcal{X}/\overrightarrow{\mathrm{d}},\ \mathcal{P}/\overrightarrow{\mathrm{p}}]$ n'est pas satisfaisable ? La réponse à cette question se décline en deux temps :

- Lors des transitions temporelles, le système peut-il rester dans un état q jusqu'à l'instant précis où il viole son invariant pour la première fois, ou doit-il l'avoir quitté avant ?
- Lors d'une transition discrète, le système doit-il vérifier l'invariant de l'état d'arrivée ?

Pour notre part, nous avons choisi de répondre "avant" au premier point et "oui" pour le second. Nous nous plaçons dans les modèle dit "de Uppaal" par Labroue [Lab98]. C'est cette interprétation qui est implantée dans des outils tels que Uppaal [LP00b] et Kronos [DOTY96, Yov97].

2.4.2.2 Exemple : exclusion mutuelle — l'algorithme du "Fisher"

Sur le même principe que le protocole du "Bakery" (utilisant des compteurs), il est possible d'utiliser des horloges. Un exemple d'un tel algorithme est l'algorithme de "Fisher" [AL91, ACD+92, STA98].

Le réseau de processus pour l'algorithme de "Fisher" est composé de n processus temporisés (identiques modulo un renommage) et manipule une variable partagée id dont le domaine de valeurs est $[0, n]$. Dans notre cas nous considérons un réseau de 2 processus.

Lorsqu'un processus veut entrer dans la section critique il doit tout d'abord être sûr que la ressource est libre ($id = 0$). Si la ressource est libre, il peut entrer dans une session de demande. Pour cela, il va passer dans un état tel qu'il aura un délai de D unité de temps pour effectuer réellement sa demande (c'est-à-dire affecter id à la valeur de son processus). Ensuite après un délai de T unité de temps, le processus regarde s'il a toujours la main sur la ressource, si c'est le cas, il obtient alors effectivement la ressource. Si ce n'est pas le cas (l'autre processus a entamé une demande et redéfini la valeur de id), il doit recommencer la session de demande.

Chaque processus peut être dans 4 états différents :

- *repos* : Le processus ne cherche pas à obtenir la ressource.
- *sessiond* : Le processus débute une session de demande de la ressource.

– *attente* : Le processus prend effectivement une option sur la ressource et attend le délai T.
– *sectionc* : Le processus a obtenu la ressource, il est en section critique.

Chaque processus fonctionne selon le schéma donné par la figure 2.3.

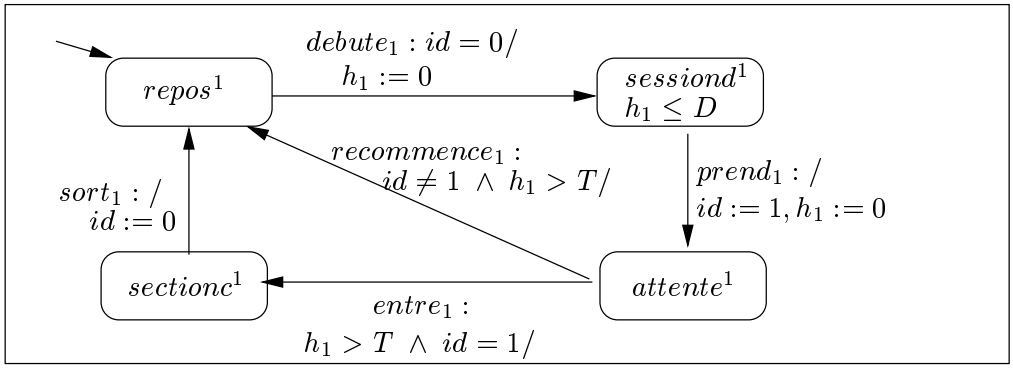


FIG. 2.3 – Description informelle du comportement d'un processus

2.4.3 Les systèmes à files d'attente avec perte

2.4.3.1 Quelques notions de langage

Nous allons tout d'abord donner quelques définitions simples sur un alphabet Σ.

Définition 14

Soit Σ un alphabet fini.
– Σ^* est l'ensemble de tous les mots finis sur Σ.
– Le mot vide est noté ε.
Soient x, $y \in \Sigma^*$.
– $x \bullet y$ est la concaténation de x et de y.
– x^n est la concaténation de n copies de x.

Définition 15 (Sous-chaîne ($\preceq$))

Soient $x = a_1 a_2 \cdots a_n$ et $y = b_1 b_2 \cdots b_m$ des mots de Σ^*.
x est une sous-chaîne de y (non nécessairement contiguë) noté $x \preceq y$ si et seulement si $\exists (i_1 \cdots i_n) \in (1 \cdots m)$ tel que $a_1 = b_{i_1}, a_2 = b_{i_2}, \ \cdots, \ a_n = b_{i_n}$.

Exemples :
$-abc \ \preceq \ adsbfc$
$-abc \ \not\preceq \ acb$

Pour deux vecteurs v=$(w_1, \cdots, w_n)$, v'=$(w'_1, \cdots, w'_n) \in (\Sigma^*)^n$, nous notons v $\preceq$ v' si et seulement si $w_i \preceq w'_i$ pour tout i $\in$ (1..n).

2.4.3.2 Définition

Dans le cadre des systèmes à files d'attente avec perte les variables sont des files d'attente qui contiennent des messages. Plus précisément le contenu des files d'attente

peut être vu comme un langage dont les lettres sont l'ensemble du vocabulaire utilisé.

Plus formellement, soit Σ l'ensemble fini des messages (vocabulaire des files), le domaine de définition de ces files est alors Σ^*.

Les opérations possibles dans ce modèle sont :
- $f_i!m_i$: la "pose" d'un message dans une file, c'est-à-dire l'*émission* d'un message dans une file.
- $f_i?m_i$: "l'enlèvement" d'un message dans une file, c'est-à-dire la *réception* d'un message dans une file.
- $vide?(f_i)$: le *test à vide* d'une file, c'est-à-dire savoir si la file contient des messages.
- nop : le "rien", c'est-à-dire une opération qui ne modifie pas le contenu des files (utile pour changer d'état de contrôle sans modifier les files).

Regardons maintenant la sémantique associée aux systèmes à files d'attente avec perte. Il nous faut définir ici la notion de relation de transition pour les systèmes à files d'attente. Une configuration est l'association d'un état de contrôle et d'un vecteur de contenu de chacune des files. Nous commençons par définir la relation de transition forte $\longrightarrow$ de la sémantique de $\mathcal{A}$. Forte, car nous ne tenons pas compte de la perte de message pour le moment. Nous étendons ensuite cette définition à la relation de transition faible qui elle tient compte de la perte des messages. Soit op un ensemble d'opérations sur les files pour une transition donnée, $Oper(op, f_i)$ retourne le type de l'opération effectuée sur la file f_i. Nous supposons qu'il n'existe qu'une seule opération sur une file donnée par transition.

Définition 16 (Relation de transition forte)
Soit $\mathcal{A} = (Q, \Sigma, \mathcal{X}, L, Op, T)$ un système à file d'attente avec perte. Soit $\mathcal{S}(\mathcal{A}) = (Q, L, \longrightarrow)$ le graphe de l'espace des configurations atteignables de $\mathcal{A}$. Pour chaque transition $t = (q_1, l, op, q_2) \in T$, nous définissons la relation de transition forte $\stackrel{l}{\longrightarrow}$ sur les configurations telle que $\forall d_1, d_2 \in (\Sigma^*)^n$, $(q_1, d_1) \stackrel{l}{\longrightarrow} (q_2, d_2)$ si et seulement si pour chaque file $f_i \in \mathcal{X}$ nous avons :
- *si $Oper(op, f_i) = f_i!m$ alors $d_2(f_i) = d_1(f_i) \bullet m$.*
- *si $Oper(op, f_i) = f_i?m$ alors $m \bullet d_2(f_i) = d_1(f_i)$.*
- *si $Oper(op, f_i) = nop$ alors $d_2(f_i) = d_1(f_i)$.*
- *si $Oper(op, f_i) = vide?(f_i)$ alors $d_1(f_i) = d_2(f_i) = \epsilon$.*

La gestion des files se fait en mode FIFO (Fisrt In First Out) : l'ajout des messages se fait en queue de file et la réception des messages se fait en tête de file.

Nous exprimons maintenant la notion de perte de message. Exprimer que les files d'attente peuvent perdre des messages, c'est dire que si une file contient par exemple la séquence abc (où $a,\ b,\ c \in \Sigma$) alors cette file contient aussi les séquences ab, ac, bc, a, b, c et ϵ. Ainsi à partir d'un contenu de file, il est possible de perdre certains messages et sur ce contenu pouvoir appliquer une transition pour aboutir à un autre contenu pouvant lui même perdre des messages. La notion de perte s'exprime sur les séquences du langages par la relation sous-chaine $\preceq$ (définition 15, page 19).

Définition 17 (Relation de transition faible)
Soit $\mathcal{A} = (Q, \Sigma, \mathcal{X}, L, Op, T)$ un système à file d'attente avec perte. Soit $\mathcal{S}(\mathcal{A}) =$

$(Q, L, \Longrightarrow)$ le graphe de l'espace des configurations atteignables de $\mathcal{A}$. Pour chaque transition $t = (q_1, l, op, q_2) \in T$, nous définissons la relation de transition faible $\overset{l}{\Longrightarrow}$ sur les configurations telle que $\forall d_1, d_2 \in (\Sigma^*)^n$, $(q_1, d_1) \overset{l}{\Longrightarrow} (q_2, d_2)$ si et seulement si $\exists d'_1,\ d'_2 \in (\Sigma^*)^n$ tels que :

$$d'_1 \preceq d_1 \ \wedge\ d_2 \preceq d'_2 \ \wedge\ (q_1, d'_1) \xrightarrow{l} (q_2, d'_2)$$

2.4.3.3 Exemple : le protocole du bit alterné

La plupart des exemples de protocoles de communication fonctionne sur le principe du modèle producteur-consommateur exprimé par la figure 2.4. Le protocole du bit alterné est le plus simple protocole de communication que l'on trouve dans la littérature tout en étant basé sur un principe repris par beaucoup d'autres protocoles.

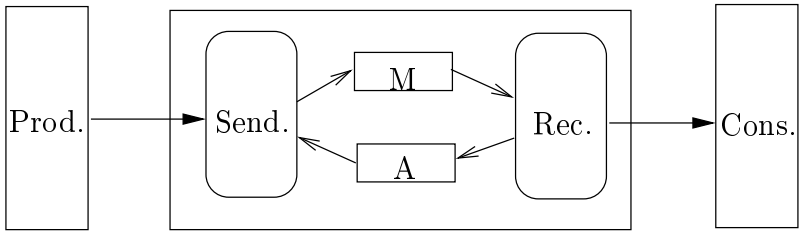


FIG. 2.4 – Modèle du producteur-consommateur

Le but de ces protocoles est d'envoyer un ensemble de messages du producteur au consommateur. Pour cela, le producteur délègue l'envoi à un émetteur qui communique via des liaisons physiques (canaux) qui ne sont pas fiables avec un récepteur chargé de transmettre les messages au consommateur. Ces liaisons peuvent perdre des messages lors de la transmission. Le producteur et le consommateur utilisent alors, par exemple, le protocole du bit alterné pour être sûr que chaque message envoyé sera reçu. C'est au protocole d'assurer la fiabilité. C'est ce que nous voulons vérifier.

Le protocole est donc constitué d'un émetteur et d'un récepteur, qui échangent des messages au travers de canaux non fiables (pouvant perdre des messages) M et A. L'émetteur lit chaque message venant du producteur (Env) qu'il transmet au récepteur. Ce dernier se chargeant de le transmettre au consommateur (Rec). Pour différencier les messages, l'émetteur associe à chacun alternativement le bit 0 ou le bit 1.

Regardons maintenant de plus près comment fonctionne le protocole. L'émetteur envoie tout d'abord le premier message avec le bit 0 (m_0) au récepteur au travers de la file M. Il attend ensuite un acquittement provenant du récepteur au travers du canal A. Tant qu'il ne reçoit pas d'acquittement pour le message m_0 (c'est-à-dire tant qu'il ne reçoit pas a_0), il réenvoie le message m_0 au récepteur au travers du canal M. En effet, ne pas recevoir l'acquittement attendu peut provenir de deux causes : soit le récepteur n'a pas reçu le message m_0 (ce dernier s'est perdu), soit le récepteur a bien reçu m_0 et a renvoyé l'acquittement a_0 mais c'est ce dernier qui a été perdu. Dans ces 2 cas, nous remarquons que les files sont supposées pouvoir contenir un nombre non borné de messages car l'émetteur doit pouvoir envoyer un nombre illimité de messages sur le canal m et symétriquement sur A de la part du récepteur.

Finalement lorsque l'émetteur reçoit l'acquittement attendu pour le message m_0, il prend le message suivant donné par le producteur et l'envoie au récepteur avec le bit 1 (m_1). Il attend maintenant l'acquittement a_1.

Et ainsi de suite, chacun des messages va être envoyé en alternant le bit de marquage à chaque changement de message.

Voici en pseudo-langage de programmation le comportement des 2 processus (émetteur et récepteur), où Envoyer(message, canal) pose "message" sur le canal et Recevoir(message, canal) enlève le premier message en tête de file et regarde s'il correspond à "message".

répéter :
  tant que non(Recevoir(a_0, A))
    Envoyer(m_0, M)
  ftq
  Recevoir(a_0, A)
  tant que non(Recevoir(a_1, A))
    Envoyer(m_1, M)
  Recevoir(a_1, A)
  ftq frépéter

L'émetteur

répéter :
  tant que non(Recevoir(m_0, M))
    Envoyer(a_0, A)
  ftq
  Recevoir(m_0, M)
  tant que non(Recevoir(m_1, M))
    Envoyer(a_1, A)
  Recevoir(m_1, M)
  ftq frépéter

Le récepteur

FIG. 2.5 – Principe de l'émetteur et du récepteur

Nous modélisons le protocole grâce au système communicant *BitAlterné* composé de deux systèmes à files *Emetteur* et *Récepteur* qui représentent respectivement le comportement de l'émetteur et du récepteur. Nous pouvons aussi écrire *BitAlterné* = *Emetteur* $\otimes$ *Récepteur*.

L'ensemble des messages manipulés par les files est $\Sigma = \{m_0, m_1, a_0, a_1\}$.

Les transitions de chacun des processus sont exprimées par les figures 2.6 et 2.7.

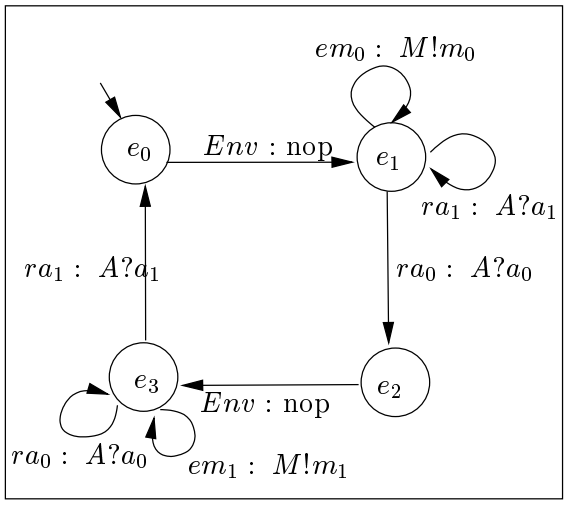


FIG. 2.6 – Modélisation de l'émetteur

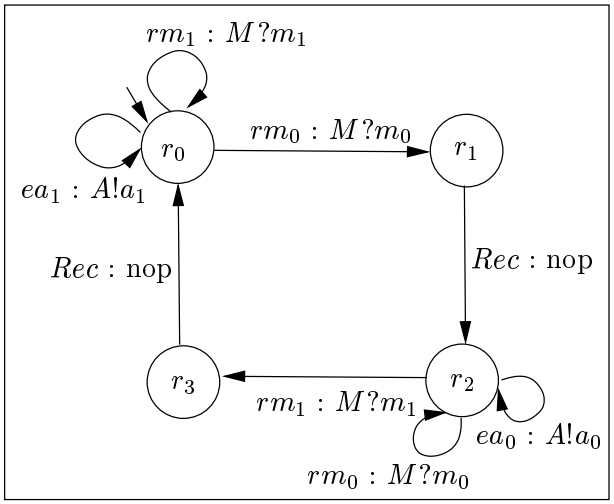


FIG. 2.7 – Modélisation du récepteur

Nous pouvons essayer de donner l'ensemble des configurations atteignables avec la relation de transition faible. La configuration initiale est $\gamma_0 = ((e_0, r_0), \epsilon, \epsilon)$: les files

sont vides. La première transition applicable est la transition de l'*Emetteur* étiquetée par Env. Nous obtenons alors la configuration $\gamma_1 = ((e_1, r_0), \epsilon, \epsilon)$. L'émetteur peut maintenant envoyer un message m_0, comme ce message peut se perdre les configurations atteignables à partir de γ_1 sont γ_1 elle-même (le message s'est perdu) ou $\gamma_2 = ((e_1, r_0), m_0, \epsilon)$ (le message est dans la file M).

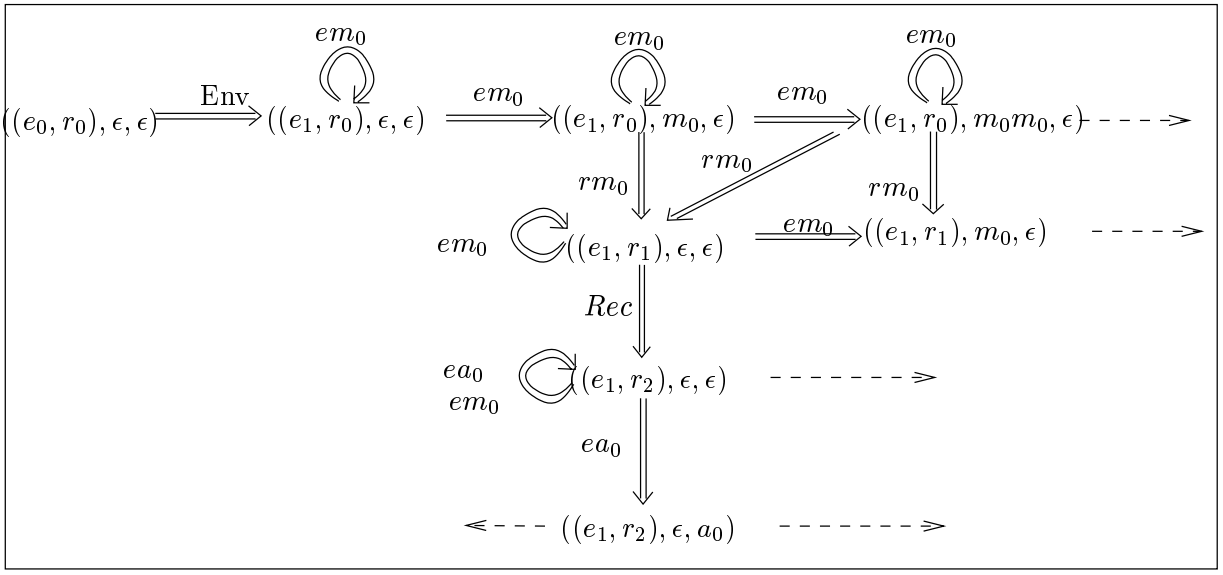


FIG. 2.8 – Ensemble des configurations atteignables (relation de transition faible)

Nous remarquons que l'ensemble des configurations est infini, par exemple la transition ajoutant m_0 dans la file peut être un nombre non borné de fois.

2.4.4 Les systèmes hétérogènes

2.4.4.1 Définition

Dans cette sorte de systèmes nous supposons pouvoir manipuler différentes variables à valeurs dans différents domaines de définitions. Nous pouvons aussi considérer que le domaine de définition de chacune des variables est l'union de tous les domaines que nous voulons considérer.

Ainsi, nous pouvons par exemple modéliser des systèmes avec des compteurs, des horloges et des files d'attente. Les opérations pour chacune de ces différentes variables sont les opérations que nous avons définies dans chacun des systèmes séparément ainsi que leur sémantique.

2.4.4.2 Exemple : le protocole de retransmission bornée — Bounded Retransmission Protocol (BRP)

Pour illustrer la notion de système hétérogène, nous prenons l'exemple du protocole de retransmission bornée.

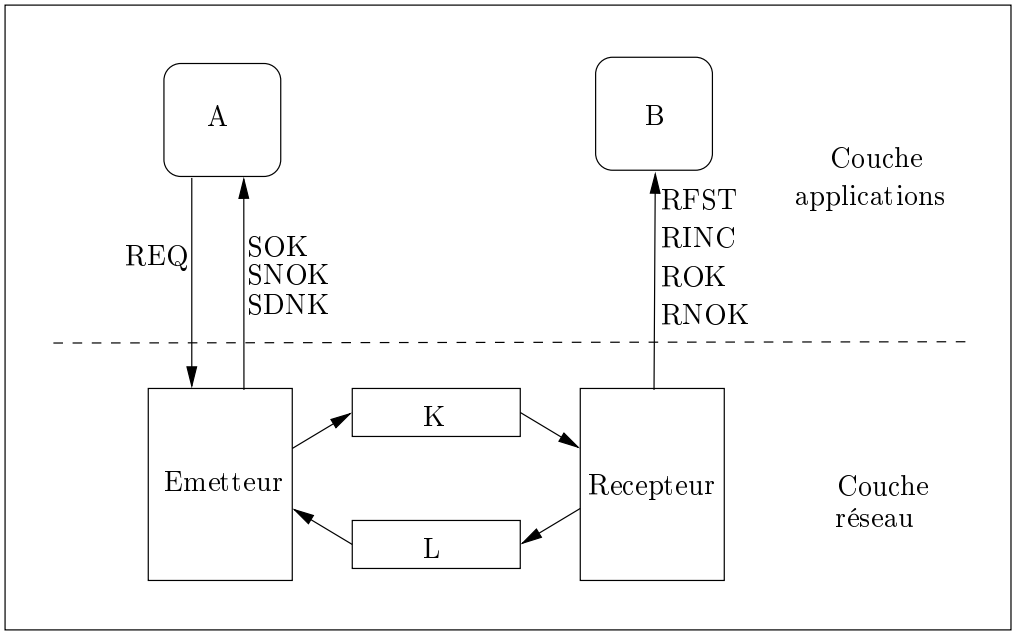


FIG. 2.9 – Protocole de retransmission bornée (BRP)

Ce protocole a été développé par le laboratoire de recherche de Philips et a été de nombreuses fois étudié [DKRT97, GS97, HS96, Mat96, HRSV01].

Le but de ce protocole est le suivant : une machine A veut envoyer un fichier de messages à travers un réseau non fiable à une machine B. Pour rendre cette transmission fiable le protocole met en œuvre, au niveau réseau, deux processus qui vont se charger de la transmission au travers de files d'attente non fiables. Durant la retransmission, chacun des processus envoie au niveau supérieur des indications sur le déroulement de l'échange. Nous détaillons ces indications dans la suite de la description.

Ces deux processus, appelés respectivement Emetteur et Récepteur, fonctionnent sur le principe de l'algorithme du protocole du bit alterné.

Regardons d'un peu plus près le fonctionnement de chacun des processus. L'émetteur reçoit (**REQ**) de A le fichier contenant les messages à envoyer à B par l'intermédiaire du récepteur. La taille du fichier (c'est-à-dire le nombre de messages à envoyer) n'est pas connue à priori, elle est donnée par un paramètre N. L'émetteur envoie les messages un par un par le canal K et attend les acquittements sur le canal L.

Après avoir envoyé un message, l'émetteur arme un time-out et attend un acquittement. Quand l'acquittement arrive, soit l'émetteur envoie le message suivant s'il existe (en changeant le bit de marquage), soit c'était le dernier message, il envoie alors au producteur un message confirmant que le fichier a bien été envoyé et reçu. Mais si le time-out expire avant que l'acquittement n'arrive, l'émetteur réenvoie le même message et réarme le time-out. La valeur du time-out est exprimée par un paramètre T_1. La réémission est répétée au plus max fois (retransmission bornée). La variable max est, elle aussi, un paramètre du système.

De son côté, le récepteur après avoir reçu un message, envoie un acquittement à l'émetteur par le canal L et donne le message reçu à B. Si ce n'était pas le dernier message, le récepteur arme un time-out et attend le message suivant. Si aucun nouveau message n'arrive avant l'expiration du time-out il conclut qu'il a perdu le contact avec l'émetteur et envoie à B un message d'erreur (signifiant que la retransmission ne s'est pas effectuée normalement).

Dans le cas particulier du dernier message, lorsque le récepteur reçoit ce dernier il arme le time-out pour renvoyer un acquittement s'il le reçoit de nouveau. A l'expiration du time-out il attend alors un message débutant une nouvelle session. La valeur du time-out est donnée par le paramètre T_2. Il se peut que l'émetteur envoie un message de début de fichier avant la fin du time-out du récepteur, ce dernier doit en tenir compte et commencer une nouvelle transmission.

Puisque le même message peut être envoyé plusieurs fois de suite, il faut pouvoir distinguer les messages les uns des autres. Nous ne détaillons pas le contenu des messages dans notre analyse nous devons donc marquer chaque message par un bit particulier (0/1) (extension du protocole du bit alterné).

Détaillons maintenant les différents messages que peut envoyer l'émetteur au producteur :

- **SOK** : la retransmission s'est effectuée sans problème, tous les messages ont été envoyés et l'émetteur a reçu un acquittement pour chacun d'eux.
- **SNOK** : la retransmission a échoué. L'émetteur n'a pas reçu d'acquittement pour un des messages. Le fichier initial n'a pas été transmis entièrement.
- **SDK** : la retransmission a peut être échoué. L'émetteur n'a pas reçu d'acquittement pour le dernier message du fichier, il ne sait pas si c'est parce que ce message n'est pas parvenu au récepteur ou si c'est l'acquittement qui a été perdu.

Quant au récepteur, il utilise les messages suivants :

- **RFST** : le récepteur a reçu le premier message d'un fichier et a initialisé une nouvelle session.
- **RINC** : le récepteur a reçu un message intermédiaire.
- **ROK** : le récepteur a reçu le dernier message. La transmission s'est bien déroulée.
- **RNOK** : le récepteur attend le message suivant mais celui-ci n'est pas arrivé avant la fin du time-out. La retransmission a échoué.

Nous remarquons que lorsque l'émetteur décide d'arrêter la retransmission il doit attendre assez longtemps pour permettre au récepteur de revenir dans son état initial (en attente du premier message).

Nous donnons les hypothèses sous lesquelles fonctionnent ce protocole, nous verrons dans les implantations comment nous avons respecté ces hypothèses :

H1 L'expiration des time-out n'est pas prématurée : un message ne peut pas arriver après que le temps ait expiré.

H2 On suppose que l'émetteur ne commence pas à retransmettre un nouveau fichier avant que le récepteur ait réagi à la panne.

Ainsi pour analyser ce protocole nous devons pouvoir manipuler des compteurs (compteurs de retransmission), des horloges (modélisant les time-out) et des files d'attente (modélisant la communication). Nous voulons analyser ce protocole en gardant la généralité des paramètres.

L'émetteur a 5 états de contrôle :
- *s_idle* : l'émetteur est en attente d'un fichier à envoyer.
- *s_next_frame* : état non stable, les transitions sortantes de cet état permettent de choisir la taille du fichier (soit le fichier n'est pas fini, le système procède alors à l'envoi d'un message intermédiaire, soit le fichier est fini, c'est l'envoi du dernier message).
- *s_wait_ack0 (1)* : état où l'émetteur attend l'acquittement d'un message ayant le bit de marquage 0 (1). Si le time-out expire, deux cas se présentent. Soit le nombre de réémissions n'est pas atteint, l'émetteur réenvoie alors le message. Soit il est atteint le système va alors dans l'état d'erreur. L'horloge est réinitialisée dans les deux cas.
- *s_error* : le système attend *synch* mesure de temps dans cet état pour permettre au récepteur de prendre en compte l'arrêt.
- *s_succes* : état non stable atteint après obtention de l'acquittement.

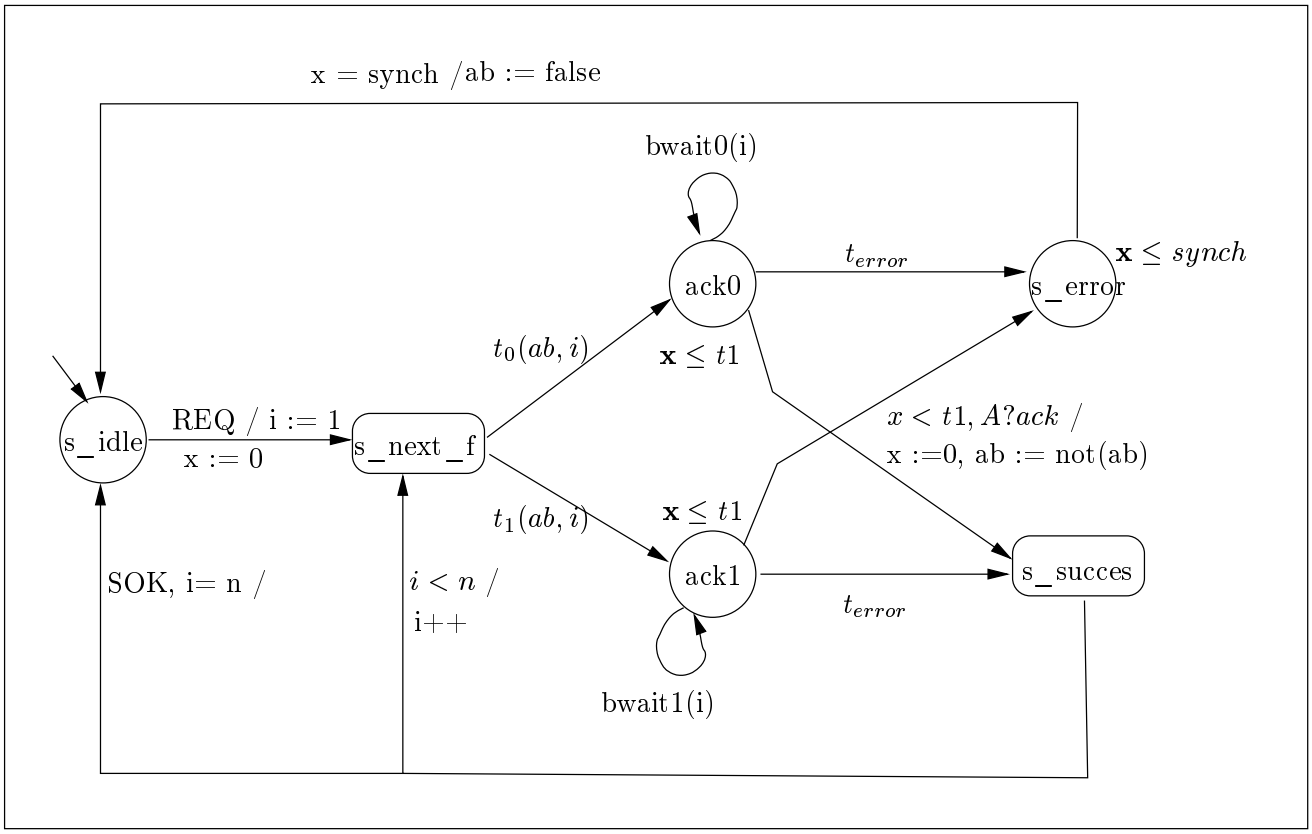


FIG. 2.10 – Brp — émetteur.

$$t_0(ab,i) = \left\{ \begin{array}{ccc} \neg ab \ \wedge \ i = 1 & / & M!fst0, \ rc := 0 \\ \neg ab \ \wedge \ 1 < i < n & / & M!m0, \ rc := 0 \\ \neg ab \ \wedge \ i = n & / & M!last0, \ rc := 0 \end{array} \right.$$

$$t_1(ab,i) = \left\{ \begin{array}{ccc} ab \ \wedge \ i = 1 & / & M!fst1, \ rc := 0 \\ ab \ \wedge \ 1 < i < n & / & M!m1, \ rc := 0 \\ ab \ \wedge \ i = n & / & M!last1, \ rc := 0 \end{array} \right.$$

$$bwait0(i) = rc < max \wedge x = t1 \wedge \left\{ \begin{array}{ccc} \neg ab \ \wedge \ i = 1 & / & M!fst0 \\ \neg ab \ \wedge \ 1 < i < n & / & M!m0 \\ \neg ab \ \wedge \ i = n & / & M!last0 \\ \neg ab \ \wedge \ /A?ack1 & & \end{array} \right\}, rc := rc{+}1, \ x := 0$$

$$bwait1(i) = rc < max \wedge x = t1 \wedge \left\{ \begin{array}{ccc} ab \ \wedge \ i = 1 & / & M!fst1 \\ ab \ \wedge \ 1 < i < n & / & M!m1 \\ ab \ \wedge \ i = n & / & M!last1 \\ ab \ \wedge \ /A?ack0 & & \end{array} \right\}, rc := rc{+}1, \ x := 0$$

$$t_{error} = \left\{ \begin{array}{l} SNOK : x = t1 \ \wedge \ rc = max \ \wedge \ i < n / x := 0 \\ SDNK : x = t1 \ \wedge \ rc = max \ \wedge \ i = n / x := 0 \end{array} \right.$$

Le récepteur contient 4 états de contrôle :
- *r_new_file* : le récepteur est en attente du premier message d'un ficher.
- *r_wait_0* : le récepteur est en attente d'un message avec le bit de marquage 0.
- *r_wait_1* : le récepteur est en attente d'un message avec le bit de marquage 1.
- *r_idle* : le récepteur attend *trr* au cas où l'acquittement ne soit pas parvenu à l'émetteur.

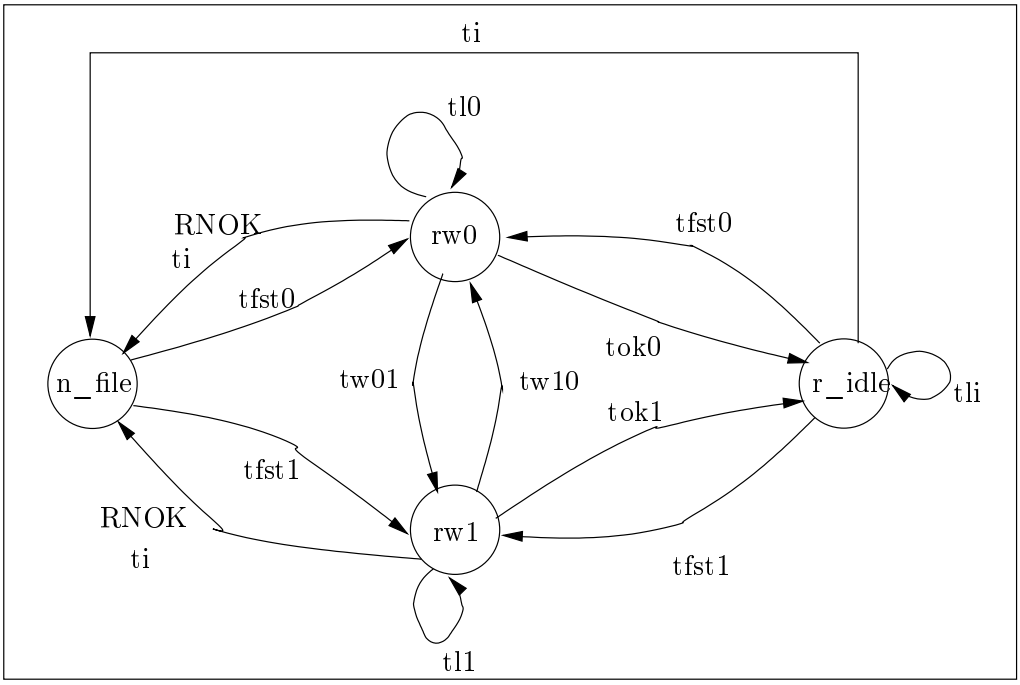


FIG. 2.11 – Brp — récepteur.

$$tfst0 = RFST \ : \ M?fst0, \ y := 0, \ A!ack0$$

$$tfst1 = RFST \ : \ M?fst1, \ y := 0, \ A!ack1$$

$$tl1 = y < trr / \left\{ \begin{array}{c} M?fst0 \\ M?m0 \end{array} \right\}, \ A!ack0$$

$$tl0 = y < trr / \left\{ \begin{array}{c} M?fst1 \\ M?m1 \end{array} \right\}, \ A!ack1$$

$$tli = \left\{ \begin{array}{l} y < trr / M?last1, A!ack1 \\ y < trr / M?last0, A!ack0 \end{array} \right.$$

$$ti = y = trr / y := 0$$

$$tw10 = RINC \ : \ y < trr / M?m1, \ A!ack1, \ y := 0$$

$$tw01 = RINC \ : \ y < trr / M?m0, \ A!ack0, \ y := 0$$

$$tok1 = ROK \ : \ y < trr / M?last1, \ A!ack1, \ y := 0$$

$$tok0 = ROK \ : \ y < trr / M?last0, \ A!ack0, \ y := 0$$

L'analyse de ce système se fait alors sous les conditions suivantes sur les paramètres (où td est les temps de transmission des messages pour aller de l'émetteur au récepteur ou vice-versa) :

$$t1 \geq 0 \ \wedge \ synch \geq 0 \ \wedge \ max \geq 2 \ \wedge \ trr \geq 0 \ \wedge$$
$$n \geq 2 \ \wedge \ t1 > 2td \ \wedge \ synch > trr \ \wedge \ trr \geq 2max * t1 + 3td \ \wedge \ td > 0$$

2.5 Discussion

Pour chacun des systèmes que nous avons définis il existe plus ou moins de résultats de décidabilité.

Les systèmes à compteurs :

Les systèmes à compteurs avec au moins deux compteurs ont le pouvoir d'expression des machines de Turing. Nous ne pouvons donc pas dans le cas général trouver un algorithme de calcul des états atteignables dont l'arrêt serait assuré pour n'importe quel type de système à compteurs. L'idée est de restreindre le champs d'application des systèmes à compteur (par exemple rester dans le cadre des formules linéaires) et trouver des méthodes partielles pour calculer l'ensemble des configurations atteignables dans certains cas particuliers.

Il existe de nombreux travaux permettant l'analyse des systèmes à compteurs dans le cadre linéaire. Comon et Jurski [CJ98] ont montré que l'ensemble des configurations atteignables d'une boucle simple d'un système à compteurs (sans paramètres) est définissable dans le cadre de la théorie additive de $\mathbb{N}$ (si nous considérons les compteurs à valeurs

dans $\mathbb{N}$). Ainsi pour les systèmes à compteurs plats (sans boucles imbriquées), l'ensemble des configurations est définissable.

Plusieurs techniques ont été proposées au cours de ces dernières années pour l'analyse des systèmes à compteurs. Nous citons celles basées sur le model-checking [DP99, DP, BW98, Boi98, FR96, Bul98] et celles basées sur l'abstraction et la création d'invariants [BBF+00].

Les systèmes à horloges :

Les automates temporisés (sans paramètres) est un modèle introduit par Alur et Dill [Dil89, AD94]. Les logiques telles que LTL, CTL, CTL* sont décidables dans le cadre de ces systèmes [Eme90, ACD90, AH92, HNSY94]. De plus [CJ99] ont montré que la relation binaire d'atteignabilité entre les configurations est définissable dans la théorie additive des réels (qui est décidable).

Des méthodes de vérification [SY96, DT98, LWYP98, MLAH99, ML99] et des outils — Kronos [DOTY96, Yov97], Uppaal [LPY97, LP00a] ou Hytech [HHWT95, HH95], par exemple, — ont été développés pour ces systèmes ces dernières années.

Pour le cas des systèmes avec paramètres [AHV93] ont montré que le problème de trouver l'ensemble des valuations des paramètres pour que le système satisfasse une propriété donnée est décidable si le modèle ne contient qu'un seul paramètre mais devient indécidable à partir de 3 paramètres. Il n'est donc pas possible dans le cas général, d'avoir une méthode automatique et totale pour tous les cas. L'analyse des systèmes paramétrés commence à être étudiée [HRSV01, AAB00].

Les systèmes à files d'attente :

De nombreuses méthodes de vérification ont été développées pour les sytèmes à files d'attente [BZ83, CF87, Pac87, BG96, BGWW97, BH97, FIS00]. Pourtant, Brand et Zafiropulo [BZ83] ont montré que tous les problèmes intéressants de vérification sur ces systèmes étaient indécidables, il n'existe donc pas, en général, de méthode complète et automatique pour cette classe de système.

Une manière de rendre le problème de la vérification décidable est de considérer les files d'attente avec perte : elles peuvent perdre des messages. Ce modèle restreint couvre tout de même une large classe de protocoles, notamment les protocoles de communication. Abdulla et Jonsson [AJ93, AJ94] ont étudié ces modèles et prouvé entre autre que le problème d'atteignabilité est un problème décidable dans ce cadre. Ils ont donné des algorithmes de vérification des propriétés de sûreté et de quelques formes de propriétés de vivacité pour ces systèmes à files d'attente avec perte. Pour cela, ils ont utilisé l'analyse en arrière : c'est-à-dire partir de l'ensemble des configurations considérées mauvaises et "remonter" les transitions (utiliser l'opérateur Pre). De cette manière, il a été prouvé que l'algorithme de recherche des configurations en arrière terminait quelque soit la configuration de départ.

Considérons l'analyse en avant :

- Cécé, Finkel et Purushotaman [CFP96] ont prouvé que l'ensemble des configurations atteignables calculé par une analyse en avant n'est pas, en général, constructible,

– et de manière plus générale, Abdulla Boasson et Bouajjani [ABB01] ont montré que quel que soit le système de réécriture à une dimension considéré il n'est pas possible d'avoir un algorithme qui construise l'ensemble des configurations atteignables pour tout ensemble de configurations de départ. Les systèmes à files d'attente peuvent effectivement être considérés comme des systèmes de réécriture, ainsi avec une file il n'est pas possible d'obtenir l'ensemble des configurations atteignables, il l'est encore moins pour n files.

Pourtant, quand cette analyse termine, nous obtenons alors une abstraction finie du modèle initial [AAB+99a]. Le principal avantage de cette abstraction est qu'elle est obtenue automatiquement, elle peut ensuite être utilisée comme un invariant pour le système.

Chapitre 3

Méthode d'analyse

Nous venons, dans le chapitre 2, de nous donner la possibilité de modéliser au travers d'automates étendus des systèmes et des algorithmes plus ou moins complexes. Nous voyons dans ce chapitre que le but n'est pas en soi de modéliser ces systèmes mais plutôt d'avoir un modèle mathématique dans lequel nous pouvons vérifier s'ils satisfont leurs spécifications. Une spécification est représentée par un ensemble de propriétés. Une propriété est l'expression d'une condition que doit satisfaire le système pour être sûr de sa correction.

De plus, nous avons introduit dans le modèle la possibilité de manipuler des paramètres. L'idée est alors de vérifier les propriétés pour un ensemble de valeurs de paramètres donné ou de trouver l'ensemble des valeurs des paramètres pour lesquelles la propriété est vérifiée. Nous appelons ce dernier point *synthèse de contraintes sur les paramètres*.

Mais tout d'abord, que faire lorsque nous disposons d'une modélisation d'un système très complexe faisant par exemple intervenir des compteurs, des horloges, des files d'attente, voire d'autres données. Souvent, il existe des méthodes développées pour telles données prises séparément mais pas réellement pour la combinaison de ces données. Il faut alors pouvoir se ramener aux cas que nous savons traiter. Cette idée est mise en œuvre par les méthodes dites d'*abstraction*. La technique employée consiste, dans un cadre mathématique bien défini, à donner une abstraction du modèle concret en étant certain que les propriétés vérifiées sur le modèle abstrait le sont également sur le modèle concret.

Ayant obtenu un système plus simple sur lequel nous pouvons agir, nous pouvons maintenant aborder le problème de la vérification de spécifications. Un automate étendu peut avoir à satisfaire deux sortes de propriétés :

- Les *propriétés de sûreté* sont des conditions sur les états et les valeurs du système telles que quel que soit le comportement du système, les valeurs et les états obtenus doivent satisfaire les conditions exprimées par la propriété. Une telle propriété est par exemple, dans le cadre d'un système d'exclusion mutuelle, l'expression de la contrainte indiquant qu'un seul processus à la fois utilise la ressource.
- Les *propriétés de vivacité* expriment une obligation de comportement pour les systèmes. Elles permettent de spécifier des conditions sur l'ordre d'exécution des tran-

sitions. Si nous reprenons l'exemple des systèmes d'exclusion mutuelle une telle propriété est la suivante : si un processus demande la ressource, inévitablement dans la suite de l'exécution du système il obtiendra cette ressource.

Nous nous intéressons dans ce chapitre à la vérification de propriétés de sûreté, nous abordons le problème de la vérification de propriété de vivacité dans le chapitre 5.

Nous voyons que le problème de la vérification de propriétés de sûreté se ramène au problème du calcul de toutes les configurations atteignables du système. En effet, il nous faut décider si toutes les configurations que peut atteindre un système satisfait bien la propriété.

Dans le cadre des systèmes dits *finis*, il existe de nombreuses méthodes pour faire ce calcul . Un système fini est tel que le graphe défini par sa sémantique est fini. Une de ces méthodes (dite méthode *énumérative*) consiste en l'énumération de toutes les configurations atteignables, ce qui est théoriquement possible car cet ensemble est fini.

Pourtant cette méthode ne s'applique pas à tous les automates étendus et cela pour deux raisons principales.

Tout d'abord, dans la plupart des exemples de systèmes à analyser, l'ensemble des configurations atteignables est infini (les valeurs que peuvent prendre les variables sont dans un domaine infini). La technique d'énumération est alors infinie, l'algorithme ne s'arrêtera pas. En effet, si nous considérons l'exemple du "bakery" (section 2.4.1.2), les compteurs peuvent prendre une infinité de valeur, aussi analyser cet exemple avec la méthode énumérative n'est pas possible.

De plus et c'est notre deuxième raison, synthétiser les contraintes sur les paramètres introduit alors des comportements infinis qui n'existaient pas pour des valuations données. En effet, si une variable est comparée à un paramètre dans une garde (par exemple $x < P / x := x + 1$) et qu'aucune contrainte ne borne ce paramètre, il sera toujours possible de prendre la transition, quelle que soit la valeur de la variable (il suffit d'ajouter la contrainte nécessaire sur le paramètre). Aussi, si la variable peut prendre une infinité de valeurs, l'ensemble des configurations sera infini.

Puisque la méthode énumérative ne peut s'appliquer dans le cadre des systèmes infinis, nous utilisons la méthode dite *symbolique*. Au lieu de manipuler des valuations de paramètres et de variables de l'automate étendu nous allons manipuler des ensembles de valuations. Pour cela, il nous faut tout d'abord choisir une représentation permettant de manipuler ces ensembles.

Calculer l'ensemble des états atteignables, nous allons le voir, c'est appliquer la relation de transition de l'automate étendu. Avoir un algorithme permettant à partir des représentations des ensembles de valuations des variables d'obtenir l'ensemble des valuations après application de la transition est donc une deuxième brique nécessaire à notre construction.

Pour aider la terminaison de l'algorithme de calcul des configurations atteignables, nous reprenons l'idée dite *d'accélération*. Le principe de l'accélération est d'obtenir en un pas l'ensemble des valuations atteignables par l'application d'un nombre infini de transitions. La troisième brique nécessaire à notre construction est donc de définir pour la structure de représentation un algorithme permettant de calculer en un pas un ensemble potentiellement infini de configurations atteignables.

Finalement, si l'on dispose de ces trois briques, nous voyons qu'il est alors possible de construire effectivement l'ensemble des configurations atteignables d'un automate donné.

Nous commençons ce chapitre en donnant quelques définitions de simulation entre systèmes de transitions étiquetés dont nous avons besoin pour la vérification.

Nous explicitons ensuite le problème de l'abstraction de systèmes.

Nous examinons, dans la partie suivante, le problème de la vérification de propriétés. Nous montrons que le problème de la vérification de propriétés de sûreté se ramène à calculer l'ensemble des états atteignables.

Avant de présenter effectivement la méthode de calcul des configurations atteignables que nous considérons nous parlons du problème de la synthèse de contraintes sur les paramètres.

Finalement nous explicitons notre méthode générale de construction de l'ensemble des configurations atteignables. Nous montrons que sous les conditions d'avoir une bonne représentation des données, une manière d'appliquer les transitions de l'automate étendu sur la représentation des ensembles de valuations et un algorithme de calcul de l'ensemble des configurations atteignables par l'application d'un nombre infini de transitions, nous avons alors un algorithme général de calcul de l'ensemble des configurations.

3.1 Préliminaires : système de transitions étiqueté (simulation)

Nous ajoutons aux définitions de systèmes de transitions étiquetés données dans la section 2.1 des notions de simulation entre deux systèmes de transitions qui vont nous être utiles dans ce chapitre.

Nous commençons par définir la relation de simulation forte $\sqsubseteq$.

Définition 18 (Simulation forte $\sqsubseteq$)

Soient $S_1 = (Q_1, L_1, \longrightarrow_1)$ *et* $S_2 = (Q_2, L_2, \longrightarrow_2)$ *deux systèmes de transitions étiquetés.*
$S_1 \sqsubseteq S_2$ *si et seulement si :*$\exists\, \mathcal{R} \subseteq Q_1 \times Q_2$ *telle que*
$\forall\ q_1 \in Q_1,\ q_2 \in Q_2,\ q_1 \mathcal{R} q_2$ *si et seulement si*
$\forall l \in L,\ \forall\ q_1' \in Q_1$ *si* $q_1 \xrightarrow{l}_1 q_1'$ *alors* $\exists\ q_2' \in Q_2$ *tel que* $q_2 \xrightarrow{l}_2 q_2'$ *et* $q_1' \mathcal{R} q_2'$

Nous allons voir qu'en général nous recherchons une simulation sur un sous-ensemble des étiquettes. Soit $obs \subseteq L$ un ensemble d'étiquettes observables. Nous définissons une simulation observationnelle entre deux STE. Nous allons tout d'abord définir la notion de système de transitions projeté sur un ensemble d'étiquettes. C'est un système dans lequel on ne garde que les transitions qui ont des étiquettes observables, les autres transitions étant cachées par la nouvelle relation de transition.

Définition 19 (Système de transition projeté)

Soit $S = (Q, L, \longrightarrow)$ *un système de transitions. Soit* $obs \subseteq L$ *un ensemble d'étiquettes observables.* $S|_{obs}$ *= (Q', obs,* $\longrightarrow_{obs}$*) est le système de transitions étiqueté projeté de S sur obs tel que :*

- $Q' \subseteq Q$.

– $\longrightarrow_{obs}$ *est définie de la manière suivante :*
$\forall\ l \in obs,\ \forall q,\ q' \in Q,\ q \xrightarrow{l}_{ops} q'$*si et seulement si* $\exists\ q'' \in Q,\ w \in (\overline{obs})^* | q \xrightarrow{w} q'' \xrightarrow{l} q'$

Les traces observationnelles, venant du système projeté, sont des sous-séquences des traces du système de transition.

Définition 20 (Simulation observationelle $\sqsubseteq_{obs}$)
Soient S = (Q, L, T) et S' = (Q', L', T') deux systèmes de transitions étiquetés. Soit $obs \subseteq L \cup L'$.

$$S \sqsubseteq_{obs} S' \text{ si et seulement si } S|_{obs} \sqsubseteq S'|_{obs}$$

Proposition 1
Soient S et S' deux systèmes de transitions.

$$S \sqsubseteq S' \Rightarrow Traces(S) \subseteq Traces(S')$$

Corollaire 1
Soient S et S' deux systèmes de transitions étiquetés.

$$S \sqsubseteq_{obs} S' \Rightarrow Traces(S|_{obs}) \subseteq Traces(S'|_{obs})$$

Proposition 2
Soient S et S' deux systèmes de transitions étiquetés.
Si $S \sqsubseteq S'$ *alors* $\forall\ obs \subseteq L\ S \sqsubseteq_{obs} S'$.

3.2 Abstraction des automates étendus

3.2.1 Le principe

L'idée de l'abstraction est introduite dans [CGL94].

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op, T)$ un automate étendu. L'abstraction est formée en obligeant les valuations des variables et des paramètres de $\mathcal{A}$ (à valeurs dans $\mathbb{D}$) à prendre leurs valeurs dans un domaine (non vide) abstrait $\hat{\mathbb{D}}$. Il faut alors donner la correspondance entre les valeurs concrètes et les valeurs abstraites. Formellement, nous supposons qu'il existe f une surjection entre les valeurs de $\mathbb{D}$ et $\hat{\mathbb{D}}$.

Nous remarquons que si le domaine $\mathbb{D}$ est un produit de domaines $\mathbb{D}_1 \times \mathbb{D}_2 \times \cdots \times \mathbb{D}_n$, nous supposons alors qu'il existe un domaine d'abstraction $\hat{\mathbb{D}}_i$ pour tout domaine $\mathbb{D}_i$ ($i \in [1, n]$). Il existe alors aussi n surjections et la fonction f est alors la suivante :

$$f((d_1, \cdots, d_n)) = (f_1(d_1), \cdots, f_n(d_n))$$

Puisque les valeurs abstraites sont dans un domaine fini et qu'elle peuvent représenter un ensemble infini, il existe une relation d'équivalence $\sim_i \subseteq \mathbb{D}_i \times \mathbb{D}_i$ définie de la manière suivante :

$$d_i \sim_i e_i \text{ si et seulement si } f_i(d_i) = f_i(e_i)$$

La surjection f induit quant à elle la relation d'équivalence $\sim \subseteq \mathbb{D} \times \mathbb{D}$ de la même manière :

$$(d_1, \cdots, d_n) \sim (e_1, \cdots, e_n) \text{ si et seulement si } d_1 \sim_1 e_1 \ \wedge \ \cdots \ \wedge \ d_n \sim_n e_n$$

Nous pouvons maintenant définir l'abstraction de l'automate $\mathcal{A}$.

Définition 21 (Abstraction)
Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op, T)$ un automate étendu, $f : \mathbb{D} \longrightarrow \hat{\mathbb{D}}$ une surjection sur les valeurs de $\mathbb{D}$ représentant une abstraction. $\hat{\mathcal{A}} = (\hat{Q}, L, \hat{T})$ est l'automate abstrait tel que :

- *$\hat{Q} = Q \times \hat{\mathbb{D}}$ est l'association de chacun des états de contrôle de $\mathcal{A}$ avec une valeur du domaine abstrait pour les variables.*
- *Soit $\hat{q}_1 = (q_1, \hat{d}_1)$, $\hat{q}_2 = (q_2, \hat{d}_2)$ deux états abstraits et l une étiquette $(\hat{q}_1, l, \hat{q}_2) \in \hat{T}$ si et seulement si*

$$\forall \ \overrightarrow{v}_1, \overrightarrow{v}_2 \in \mathbb{D}^{n_v}. f(\overrightarrow{v}_1) = \hat{d}_1 \wedge f(\overrightarrow{v}_2) = \hat{d}_2 \wedge \exists (q_1, l, op, q_2) \in T. (q_1, \overrightarrow{v}_1) \xrightarrow{l} (q_2, \overrightarrow{v}_2)$$

Remarque : Dans le cadre du domaine définition représentant un produit de sous domaine, il est possible de n'abstraire qu'une partie de ces sous-domaines. Dans ce cas, l'automate abstrait est composé des états produit des états de contrôle de $\mathcal{A}$ et des valeurs abstraites, des variables et des paramètres dont le domaine n'est pas abstrait et des transitions définies comme expliqué dans la définition 21 où nous conservons les opérations sur les variables non abstraites.

Cette méthode est exacte pour toute propriété exprimée sous la logique CTL^* [CGL94]. Nous pouvons donc vérifier des propriétés de sûreté sur le modèle abstrait, il est certain qu'elles seront vérifiées si et seulement si elles pouvaient l'être dans le modèle concret.

Il existe des outils tels que InVeST [BLO98] permettant d'obtenir automatiquement l'automate abstrait à partir d'un modèle concret et d'une fonction d'abstraction. Le problème est alors de trouver la bonne fonction d'abstraction.

3.2.2 Un exemple : le BRP

Reprenons maintenant l'exemple du protocole de retransmission bornée (BRP) que nous avons décrit page 23. Le système proposé manipule des files d'attente, des compteurs, des horloges et des booléens. Le modèle complexe complet est très difficile à analyser, aussi avant de vérifier des propriétés nous nous ramenons à des cas plus simples (que nous savons traiter). Dans ce cas particulier nous pouvons par exemple commencer par abstraire les horloges et les compteurs afin d'obtenir un système ne contenant que des files d'attente (cf. figure 3.1 et 3.2).

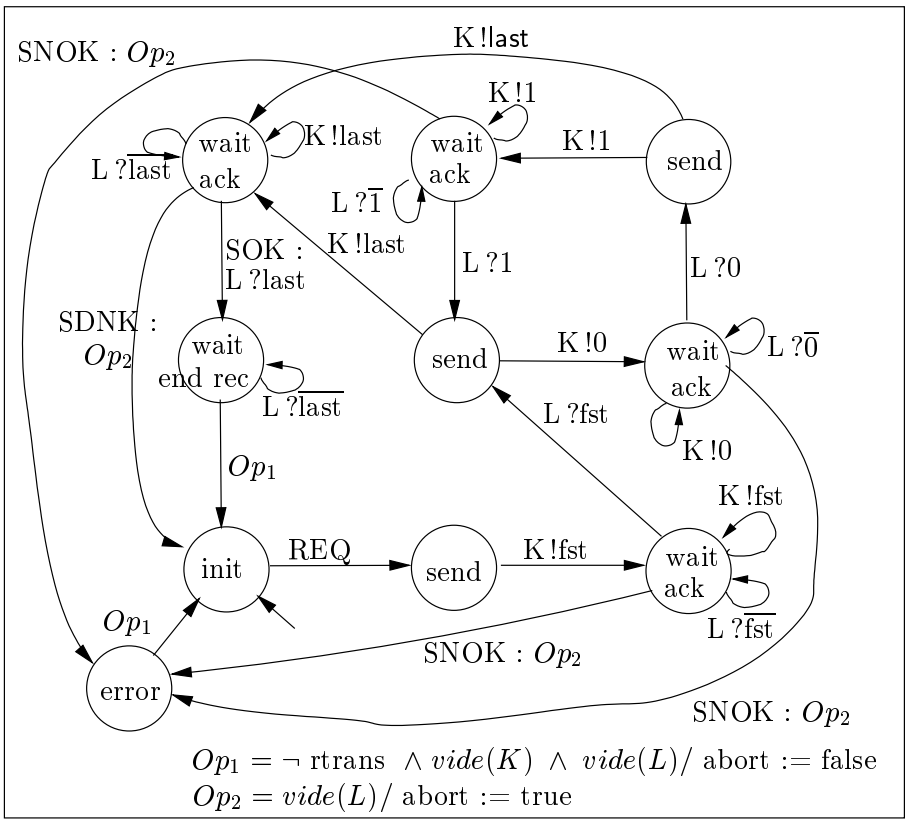


FIG. 3.1 – Abstraction 1 : l'émetteur

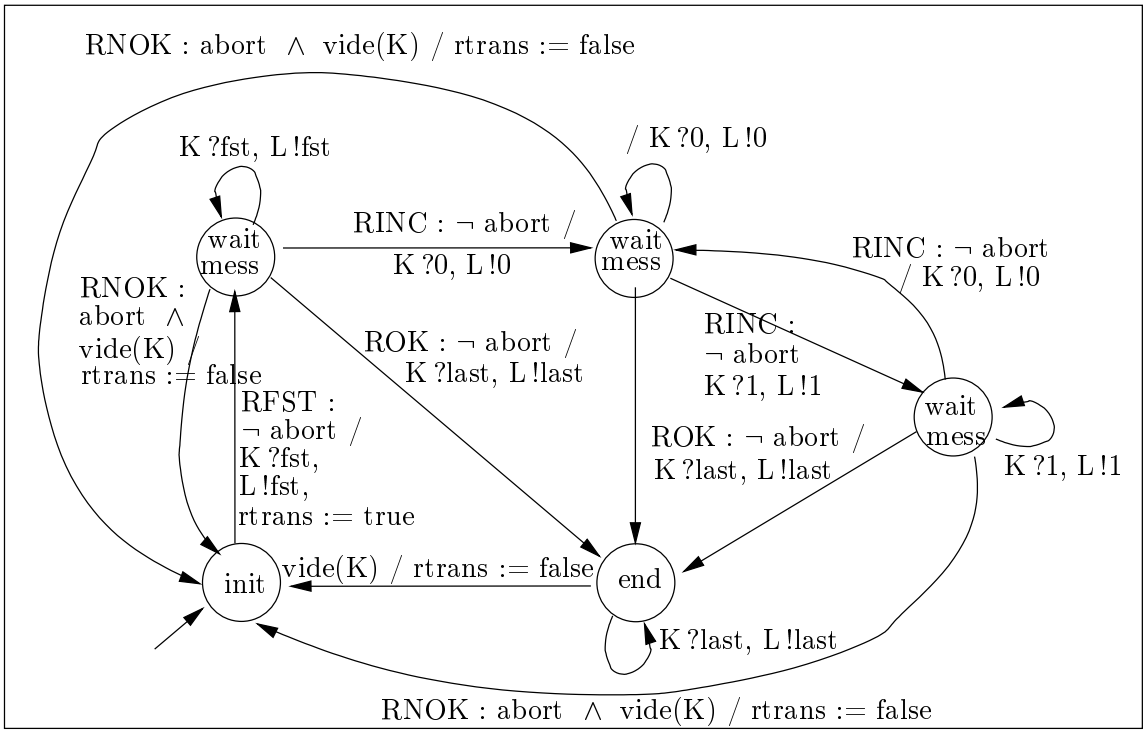


FIG. 3.2 – Abstraction 1 : le récepteur

3.3 Vérification de propriétés de sûreté

Nous voulons savoir si un système satisfait une propriété donnée. Nous considérons ici le cas des propriétés linéaires : dont la vérification peut se faire sur les traces. Ainsi, à la description du comportement d'un système, s'ajoute la notion de correction. La correction est exprimée par une suite de propriétés que doit satisfaire le système.

Il existe de nombreuses manières d'exprimer ces propriétés : du langage naturel aux représentations en logique. Puisqu'une propriété de sûreté exprime une condition sur les configurations et sur l'ordre d'exécution des transitions du système, quel que soit le formalisme utilisé pour représenter les propriétés, il faut trouver un moyen de décider si cette propriété est satisfaite pour toutes les configurations atteintes par le système.

Si l'on se souvient que l'ensemble des configurations atteignables et le comportement d'un système est représenté par le graphe de l'espace d'état, nous montrons alors qu'il existe un algorithme permettant de décider de la satisfaction de la propriété "à la volée" si l'on exprime cette propriété sous forme de système de transition. L'idée de la vérification à la volée est d'observer la satisfaction de la propriété tout en calculant l'ensemble des configurations atteignables. Ainsi, lorsque l'algorithme de calcul des configurations atteignables termine, la propriété est vérifiée. Si l'ensemble des configurations atteignables est infini, la vérification à la volée permet de détecter une non satisfaction de la propriété sans attendre la fin du calcul (qui de plus ne se termine pas !).

Dans ce travail nous avons choisi de représenter une propriété par un système de transitions étiqueté.

Il existe aussi de nombreuses autres représentations telles que :

- *la logique linéaire temporelle (Linear Temporal Logic : LTL)* qui fut introduite par Pnueli [Pnu77] comme langage de description des séquences d'exécution (ou séquence d'états). LTL est composée d'une logique décrivant les états — qui peut être propositionnelle ou de 1^{er} ordre — et d'opérateurs temporels qui spécifient sur quelle séquence d'états la formule doit être vérifiée. Il a été montré que la logique temporelle propositionnelle est décidable. Cette logique a été utilisée pour le model-checking (vérification de propriétés sur un modèle fini) [QS82, CES86].
- *la logique linéaire temporelle contrainte (Constraint Linear Temporal Logic : CTL)* [Eme90] permet d'exprimer des propriétés sur les chemins du graphe symbolique.

Nous montrons que le problème de la vérification de propriétés de sûreté se ramène dans notre cas à un problème d'atteignabilité dans le graphe de l'espace des états.

Une propriété de sûreté contient deux parties. D'une part elle exprime une condition sur les états de contrôle de l'automate étendu, sur les valeurs des variables obtenues au cours de l'analyse. D'autre part, elle exprime l'attente d'un comportement prédéfini. Un comportement est prédéfini si l'on peut prédire (ou imposer) l'ordre dans lequel le système doit exécuter ses transitions.

Dans notre cadre, le comportement attendu est représenté par une suite de transitions du système de transitions étiqueté. Généralement, la propriété ne concerne qu'un sous-ensemble des étiquettes de l'automate étendu modélisant le système. Nous parlons alors d'*ensemble des étiquettes observables*, noté Ops, qui est un sous-ensemble des étiquettes

de l'automate étendu. Ce sont les étiquettes mises-en-œuvre dans la propriété.

Pour ce qui est des conditions sur les états de contrôle et sur les valeurs des variables, nous supposons qu'il existe une logique permettant de représenter ces contraintes. Nous associons alors à chaque état du système de transitions étiqueté représentant la propriété une formule représentant la contrainte à satisfaire. Par exemple, si les variables du modèle sont des compteurs, nous pouvons utiliser des contraintes de l'arithmétique de Presburger. Par analogie avec les logiques, quelles ques soient les données du modèle, nous supposons qu'il existe une représentation permettant d'écrire ces contraintes que nous notons $F(\mathcal{X})$ tel que la notion de satisfaisabilité (savoir en fonction d'une valuation des variables donnée si la formule est vraie) est décidable. Il existe alors une fonction $sat(\phi, \overrightarrow{d})$ où $\phi \in F(\mathcal{X})$ exprime une condition sur les variables et $\overrightarrow{d}$ est une valuation de ces variables, exprimant la satisfaisabilité.

Une propriété est donc définie de la manière suivante.

Définition 22 (Propriété de sûreté)

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op, T)$ un automate étendu. Soit $Obs \subseteq L$ l'ensemble des étiquettes observables. Une propriété de sûreté sur l'automate $\mathcal{A}$ est le couple $(\mathcal{G}, V)$ où :

- *$\mathcal{G} = (Q', Obs, \longrightarrow)$ est un système de transitions étiqueté.*
- *$V : Q' \longrightarrow F(\mathcal{X} \cup \mathcal{P})$ est une fonction qui associe à chaque état de $\mathcal{G}$ une condition sur les variables et les paramètres de $\mathcal{A}$.*

La question qui se pose à nous maintenant est de savoir si l'automate étendu $\mathcal{A}$ vérifie la propriété de sûreté.

Puisque les valeurs des variables interviennent dans la propriété, nous devons être capables de tester pour chaque configuration obtenue si elle satisfait la condition exprimée par V. Nous allons définir une opération de composition entre l'automate étendu $\mathcal{A}$ et le graphe de propriété $\mathcal{G}$ pour obtenir un nouvel automate étendu $\mathcal{A}_{\mathcal{P}}$ sur lequel nous vérifions la propriété. Les états de contrôle de $\mathcal{A}_{\mathcal{P}}$ sont l'association d'un état de $\mathcal{A}$ et d'un état de $\mathcal{G}$. Les transitions de l'automate composé $\mathcal{A}_{\mathcal{P}}$ sont le résultat d'un produit synchrone entre les transitions $\mathcal{A}$ et $\mathcal{G}$ pour les transitions portant une étiquette observable et d'un produit asynchrone pour les transitions ayant des étiquettes non observables.

En effet, pour que nous puissions décider de la propriété il faut vérifier que le comportement prédéfini par la propriété est bien simulé par l'automate $\mathcal{A}$. Ainsi, nous synchronisons les transitions contenant des étiquettes observables.

Mais que se passe-t-il si à partir d'une configuration donnée, $\mathcal{A}$ veut prendre une transition étiquetée par un label observable et que l'automate représentant la propriété n'a pas de transition étiquetée par ce même label ? En fait, pour ne jamais avoir ce cas nous allons compléter le graphe $\mathcal{G}$ de telle sorte que :

- Pour toute étiquette observable l, il existe à partir de tout état de $\mathcal{G}$ une transition étiquetée l. Pour cela, nous ajoutons un état nommé $Trappe$ à l'ensemble des états de $\mathcal{G}$ et nous ajoutons pour chaque état q (différent de $Trappe$) les transitions manquantes suivantes : $(q, l, Trappe)$ telles que l est une étiquette observable qui n'existait pas sur les transitions sortant de l'état q.
- De plus, nous modifions la sémantique des automates étendus pour prendre en compte les conditions sur les variables. Si ces conditions ne sont pas vérifiées l'état $Trappe$ est alors atteignable.

Nous commençons par la syntaxe de l'opération de composition $\otimes$.

Définition 23 (Composition $\otimes$)

Soient $\mathcal{A} = (Q_1, \mathcal{X}_1, \mathcal{P}_1, L_1, Op_1, T_1)$ un automate étendu, $\mathcal{G} = (Q_2, Obs, T_2)$ un système de transitions étiquetée complet tel que $Obs \subseteq L_1$. $\mathcal{A}_\mathcal{P} = (Q, \mathcal{X}, \mathcal{P}, L, Op, T) = \mathcal{A} \otimes \mathcal{G}$ est l'automate étendu résultant de la composition si :

- *$Q = Q_1 \times Q_2$ est l'ensemble des états de contrôle. Un état de contrôle $q \in Q$ est la composition d'un état de contrôle provenant de $\mathcal{A}$ et d'un état de contrôle provenant de $\mathcal{G}$.*
- *$\mathcal{X} = \mathcal{X}_0$ est l'ensemble des variables.*
- *$\mathcal{P} = \mathcal{P}_0$ est l'ensemble des paramètres.*
- *$L = L_1$ est l'ensemble de toutes les étiquettes.*
- *$Op = Op_1$ est l'ensemble des opérations possibles.*
- *$T \subseteq Q \times L \times Op \times Q$ est une nouvelle relation de transition définie par :*

$$\frac{l \notin Obs,\ q_1 \in Q_1,\ q_2 \in Q_2,\ (q_1, l, op, q_1') \in T_1}{((q_1, q_2), l, op, (q_1', q_2)) \in T}$$

$$\frac{l \in Obs,\ q_1 \in Q_1,\ q_2 \in Q_2,\ (q_1, l, op, q_1') \in T_1\ \ (q_2, l, q_2') \in T_2}{((q_1, q_2), l, op, (q_1', q_2')) \in T}$$

Nous avons maintenant un automate étendu $\mathcal{A}_\mathcal{P}$ dont les états de contrôle sont composés d'un état de l'automate initial $\mathcal{A}$ et d'un état du système de transition représentant la propriété $\mathcal{G}$. Soit $(\mathcal{G}, V)$ une propriété, on note par $V(q)$ la condition sur les variables associées à l'état q de $\mathcal{G}$. Soient $\mathcal{A} = (Q_1, \mathcal{X}_1, \mathcal{P}_1, L_1, Op_1, T_1)$ un automate étendu, $\mathcal{G} = (Q_2, Obs, T_2)$ un système de transitions étiqueté complet tels que $Obs \subseteq L_1$ et $\mathcal{A}_\mathcal{P} = (Q, \mathcal{X}, \mathcal{P}, L, Op, T) = \mathcal{A} \otimes \mathcal{G}$ l'automate composé correspondant. Une configuration γ de l'automate composé est telle que $\gamma = ((q_1, q_2), \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$ où q_1 est un état de contrôle de l'automate étendu initial, q_2 est un état de contrôle de $\mathcal{G}$, $\overrightarrow{\mathrm{d}}$ est une valuation sur les variables et $\overrightarrow{\mathrm{p}}$ est une valuation sur les paramètres. Nous redéfinissons la sémantique de l'application d'une transition de la manière suivante :

$(\ ((q_1, q_2), \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}), l, ((q_1', q_2'), \overrightarrow{\mathrm{d}'}, \overrightarrow{\mathrm{p}'})\ )$ est une transition de la sémantique de l'automate composé $\mathcal{S}(\mathcal{A}_\mathcal{P})$ si et seulement si

- $l \notin Ops$: $(q_1, l, op, q_1') \in T_1$
  $\overrightarrow{\mathrm{d}'} = op(\overrightarrow{\mathrm{d}})$
  $\overrightarrow{\mathrm{p}'} = \overrightarrow{\mathrm{p}}$
  si $sat(V(q_2), \overrightarrow{\mathrm{d}'}, \overrightarrow{\mathrm{p}'})$ alors $q_2' = q_2$ sinon $q_2' = Trappe$.
- $l \in Ops$: $(q_1, l, op, q_1') \in T_1$ et $(q_2, l, q_2') \in T_2$
  $\overrightarrow{\mathrm{d}'} = op(\overrightarrow{\mathrm{d}})$
  $\overrightarrow{\mathrm{p}'} = \overrightarrow{\mathrm{p}}$
  si non$(sat(V(q_2'), \overrightarrow{\mathrm{d}'}, \overrightarrow{\mathrm{p}'}))$ alors $q_2' = Trappe$.

Nous pouvons alors calculer l'ensemble des configurations atteignables de $\mathcal{A}_\mathcal{P}$. Le problème de la vérification de la propriété se ramène alors à vérifier qu'aucune configuration atteignable ne contient l'état $Trappe$. Aussi, si l'équation suivante est vérifiée, l'automate $\mathcal{A}$ satisfait bien la propriété.

Proposition 3

Soient $\mathcal{A} = (Q_1, \mathcal{X}_1, \mathcal{P}_1, L_1, T_1)$ un automate étendu, $(\mathcal{G} = (Q_2, Obs, T_2), V)$ une propriété de sûreté tels que $Ops \subseteq L_1$ et $\mathcal{A}_\mathcal{P} = (Q, \mathcal{X}, \mathcal{P}, L, T) = \mathcal{A} \otimes \mathcal{G}$ l'automate étendu composé. Soit $((q_{1,0}, q_{2,0}), \overrightarrow{d}_{x0}, \overrightarrow{d}_{p0})$ la configuration initiale du système telle que $q_{1,0} \in Q_1$ et $q_{2,0} \in Q_2$. Si

$$\forall\ \gamma = ((q_1, q_2), \overrightarrow{d}, \overrightarrow{p}) \in Post^*(((q_{1,0}, q_{2,0}), \overrightarrow{d}_0, \overrightarrow{p}_0)).\ q_2 \neq Trappe$$

alors la propriété est vérifiée.

Ainsi, pour vérifier une propriété de sûreté exprimée par un système de transitions étiqueté, nous devons alors calculer l'ensemble des configurations atteignables de l'automate étendu $\mathcal{A}_\mathcal{P}$.

3.4 Synthèse de contraintes sur les paramètres

Jusqu'à présent, nous avons discuté de la vérification de propriétés à partir de configurations contenant des valuations de paramètres. Si l'on se souvient que les opérations mises en œuvre dans les transitions ne changent pas les valeurs des paramètres, on se rend compte que l'analyse d'un système se fait donc pour des valeurs données initiales. Si nous voulons changer ces valeurs — par exemple, imaginons avoir vérifié le comportement d'un ascenceur dans un immeuble de 10 étages et nous voudrions maintenant refaire l'analyse pour 11 étages — il faut recalculer l'ensemble des configurations atteignables avec ces nouvelles valeurs et cela pour chaque changement.

De plus nous vérifions alors le système pour un certain nombre de valeurs différentes des paramètres mais rien ne permet d'être convaincu que le système satisfait la propriété pour toute valeur des paramètres. Dans certains cas particuliers, il est possible de déduire à la main les contraintes liant les paramètres telles que pour toute valuation satisfaisant ces contraintes le système satisfait la propriété. Pourtant, il n'est pas aisé de trouver ces contraintes. De plus, cette génération de contraintes faite "à la main" est généralement source d'erreurs.

L'idée de la synthèse automatique de contraintes sur les paramètres est alors de caractériser automatiquement (sans intervention humaine) l'ensemble des valuations des paramètres tel que pour toute valuation de cet ensemble le système analysé avec cette valeur satisfait la propriété donnée. Soit $\mathcal{A}$ un automate étendu, soit $\mathcal{S}(\mathcal{A})$ sa sémantique, soit $\overrightarrow{\mathrm{p}}$ une valuation sur les paramètres de $\mathcal{A}$, on note par $\mathcal{S}(\mathcal{A})_{\overrightarrow{\mathrm{p}}}$ le graphe de l'espace des états calculé avec la valuation $\overrightarrow{\mathrm{p}}$ sur les paramètres. Soit $\mathcal{G}$ une propriété de sûreté que doit satisfaire le système. Nous cherchons à calculer l'ensemble de valuations des paramètres suivant :

$$\{\overrightarrow{\mathrm{p}} \mid \mathcal{S}(\mathcal{A})_{\overrightarrow{\mathrm{p}}} \models \mathcal{G}\}$$

Il est aussi possible de calculer l'ensemble des configurations atteignables d'un automate étendu, sans donner de valeur à priori pour les paramètres. Nous obtenons alors l'ensemble des comportements possibles pour un automate donné.

3.5 Construction de l'ensemble des configurations atteignables

Nous venons de voir que pour vérifier une propriété ou pour synthétiser les contraintes sur les paramètres, nous devons calculer l'ensemble des configurations atteignables.

Nous commençons par la présentation d'une méthode de calcul pour les systèmes finis puis nous traitons le cas des systèmes infinis.

3.5.1 Systèmes finis

Calculer l'ensemble des configurations atteignables d'un automate étendu $\mathcal{A}$ est simple lorsque cet ensemble est fini. Il suffit alors de commencer le calcul avec un ensemble ne contenant que la configuration initiale. Puis d'appliquer itérativement la relation d'atteignabilité en un pas R de l'automate en ajoutant à l'ensemble toutes les configurations obtenues par cette relation. Le nombre de configurations étant fini, l'ensemble des configurations calculé va se stabiliser (i.e. toutes les configurations qui pourront être obtenues par la relation d'atteignabilité appartiennent déjà à l'ensemble calculé). Lorsque la stabilité est atteinte, l'ensemble calculé représente alors exactement l'ensemble des configurations atteignables de l'automate.

Si l'on se souvient que l'espace des configurations atteignables d'un automate étendu est un graphe dont les noeuds sont les configurations atteignables et les arcs correspondent à l'application de la relation d'atteignabilité en un pas, on peut considérer que la méthode de calcul décrite dans le paragraphe précédent peut être vu comme une *exploration* du graphe de l'espace des configurations. Il existe plusieurs algorithmes permettant l'exploration d'un graphe, ils ne se différencient que par l'ordre dans lequel les nœuds sont visités. Pour notre part, nous nous intéressons à l'algorithme dit *en profondeur d'abord*. Cette méthode est illustrée par l'algorithme donné à la figure 3.3 : la stratégie de cet algorithme étant de toujours suivre une transition dont l'origine est le nœud le plus récemment visité. Cet algorithme met en œuvre l'ensemble de configurations *états_visités* qui représente l'ensemble des états que l'algorithme a déjà visité.

fonction EnProfondeur (AMSP $(\mathcal{A} = (Q,\ \mathcal{X},\ \mathcal{P},\ L,\ Op,\ T))$) :
ensemble de configurations
lexique :
états_visités : ensemble de configurations

procédure explore (configuration $\gamma = (q,\ \overrightarrow{\mathrm{d}},\ \overrightarrow{\mathrm{p}})$)
algo. :
états_visités := *états_visités* $\cup \{(q,\ \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})\}$
pour toutes transitions $(q,\ l,\ op,\ q') \in T$ **faire**
$\overrightarrow{\mathrm{d}}' = op(\overrightarrow{\mathrm{d}})$
si ($(\overrightarrow{\mathrm{d}}' \neq \bot)$ **et puis** $((q',\ \overrightarrow{\mathrm{d}}',\ \overrightarrow{\mathrm{p}}) \notin$ *états_visités*))
alors explore $((q',\ \overrightarrow{\mathrm{d}}',\ \overrightarrow{\mathrm{p}})$)
finalgo.

algo. :
états_visités $= \emptyset$
explore$((q_0,\ \overrightarrow{\mathrm{d}}_0,\ \overrightarrow{\mathrm{p}}_0))$
retourner *états_visités*
finalgo.

FIG. 3.3 – Exploration en profondeur de l'espace des configurations

Ainsi, nous avons un algorithme permettant de calculer l'ensemble des configurations atteignables d'un automate étendu lorsque cet ensemble est fini. Pourtant nous avons vu dans le chapitre précédent (section 2.4) que cet ensemble n'est pas forcément fini.

3.5.2 Systèmes infinis

Nous avons vu dans le chapitre précédent que l'ensemble des configurations atteignables peut être infini. Nous commençons cette partie en donnant les différentes caractéristiques que doit suivre la représentation symbolique choisie pour manipuler des ensembles de valuations de variables et de valuations de paramètres.

3.5.2.1 Représentation symbolique

Nous voulons avoir une représentation permettant de manipuler des ensembles de valuations de variables et aussi des ensembles de valuations de paramètres (pour pouvoir faire de la synthèse). En fait, il n'est pas possible en général de représenter l'ensemble des sous-parties de $2^{\mathbb{D}^{n_x+n_p}}$ où nous rappelons que $\mathbb{D}$ est le domaine de définition des variables (et des paramètres), n_x est le nombre de variables et n_p le nombre de paramètres.

Les fondations du calcul des configurations atteignables de manière symbolique reposent sur le fait de représenter (de manière à avoir de bonnes propriétés pour pouvoir les manipuler) des ensembles de valuations pour les variables et pour les paramètres.

Supposons que nous sachions caractériser l'ensemble des ensembles de valuations accessibles lors de l'analyse des systèmes que nous considérons. Nous dénotons cet ensemble $E^{\mathbb{D}} \subseteq 2^{\mathbb{D}^{n_x+n_p}}$. Supposons que les objets (que nous allons manipuler effectivement dans les algorithmes) représentant ces ensembles de valuations appartiennent à l'ensemble *Rep*. Nous définissons alors une fonction de représentation *rep* qui associe à chaque ensemble de $E^{\mathbb{D}}$ un objet de l'ensemble de représentation et, de manière symétrique, nous définissons la fonction *ens* qui permet à partir d'un objet de la représentation de donner l'ensemble qu'il représente. La définition suivante permet de formaliser ces fonctions.

Définition 24 (Représentation symbolique)
Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, Op, L, T)$ un automate étendu tel que l'ensemble des variables $\mathcal{X}$ contient n_x variables sur le domaine $\mathbb{D}$ et l'ensemble des paramètres $\mathcal{P}$ contient n_p variables sur ce même domaine $\mathbb{D}$. Nous définissons la fonction rep et ens de la manière suivante :

- $rep : E^{\mathbb{D}} \longrightarrow Rep$
- $ens : Rep \longrightarrow E^{\mathbb{D}}$

Pour pouvoir utiliser les objets de la représentation dans notre construction, ils doivent satisfaire certaines "bonnes" propriétés (propriétés explicitées ci-après).

Propriété 1
Soit $\mathcal{A} = (Q,\ \mathcal{X},\ \mathcal{P},\ L,\ Op,\ T)$ un automate étendu. Soit $E^{\mathbb{D}} \subseteq 2^{\mathbb{D}^{n_x+n_p}}$ l'ensemble des valuations accessibles au cours de l'analyse de l'automate étendu. Soit Rep un ensemble d'objets de représentation sur lequel nous avons défini les fonctions rep et ens. La représentation symbolique est une bonne représentation *si et seulement si*

- *Pour tout ensemble de valuations atteignables de variables et de paramètres de l'automate il existe un objet pour le représenter :*

$$\forall X \in E^{\mathbb{D}}\ \exists M \in Rep\ .\ rep(X) = M$$

  En particulier, il existe une représentation de l'ensemble vide et une représentation de l'ensemble de toutes les valuations possibles (ou ensemble universel).
- *Les éléments de l'ensemble Rep sont regroupés en classe d'équivalence de la manière suivante ($\mathcal{C}(M)$ est l'ensemble des éléments de la représentation appartenant à la classe d'équivalence de M) :*

$$\mathcal{C}(M_1) = \{M \mid \exists X \in E^{\mathbb{D}}\ ens(M_1) = X\ \wedge\ ens(M) = X\}$$

- *Pour chaque classe d'équivalence on distingue un élément que l'on appelle* élément canonique *qui est le représentant de tous les objets de sa classe. Il est unique.*
- *Soit une opération $op \in \{\cup,\ \cap\}$, soit X le résultat de la composition de deux ensembles de $E^{\mathbb{D}}$ par op, il existe alors une opération sur les ensembles de représentation op_r ($\cup_r$ pour $\cup$ et $\cap_r$ pour $\cap$) telle qu'il soit alors possible à partir de deux représentations des ensembles initiaux d'obtenir un troisième objet de représentation M tel que $ens(M) = X$:*

$$\forall X_1, X_2 \in E^{\mathbb{D}}\ \forall op \in \{\cup,\ \cap\}\ \forall M_1, M_2 \in Rep\ \forall X \in E^{\mathbb{D}}\ \exists M \in Rep.$$

$$rep(X_1) = M_1 \wedge rep(X_2) = M_2 \wedge X_1\ op\ X_2 = X \Rightarrow M_1\ op_r\ M_2 = M \wedge ens(M) = X$$

– *Il existe une opération d'inclusion entre les représentations telle que :*

$$\forall X_1, X_2 \in E^{\mathbb{ID}}\ \forall M_1, M_2 \in Rep.$$

$$M_1 = rep(X_1)\ \wedge\ M_2 = rep(X_2)\ \wedge\ X_1 \subseteq X_2 \Rightarrow M_1 \subseteq_r M_2$$

– *Le test du vide doit être décidable pour la représentation :*

$$\textit{le test } ens(M) = \emptyset\ ?\ \textit{doit être décidable pour la classe de représentation.}$$

Nous verrons que, dans certains cas, nous relacherons la propriété sur l'union de deux objets de la représentation. En effet, l'union nous permet de limiter la taille des objets sauvegardés par l'algorithme mais n'est pas nécessaire.

3.5.2.2 *Post* effectivement constructible

Nous avons vu au paragraphe 3.5.1 que l'algorithme permettant la construction de l'ensemble des configurations atteignables était un parcours de graphe en profondeur. Pour construire ce graphe, il faut pouvoir appliquer les transitions. Par contre, pour pouvoir continuer à manipuler les objets de représentation il faut que l'application d'une transition sur un objet de la représentation préserve la classe d'ensemble, c'est-à-dire le résultat de l'application doit appartenir à $E^{\mathbb{ID}}$.

Nous ajoutons donc quelques propriétés à l'ensemble de représentation *Rep*.

Propriété 2 (*Post* effectivement constructible)
Soit $\mathcal{A} = (Q,\ \mathcal{X},\ \mathcal{P},\ L,\ Op,\ T)$ un automate étendu. Pour chaque transition de l'automate (donc pour chaque opération) il existe une manière de calculer l'ensemble des configurations atteignables — il existe un algorithme permettant de le calculer appelé $post_t$ — par cette transition à partir d'un ensemble de valuations des variables et des paramètres X donné :

$$\forall X, X' \in E^{\mathbb{ID}}\ \forall M_1, M_2 \in Rep.\ M_1 = rep(X)\ \wedge\ (M_2) = post_t(M_1) \Rightarrow ens(M_2) = X'$$

où $X' = \{(\overrightarrow{d}', \overrightarrow{p}') \mid \forall(\overrightarrow{d}, \overrightarrow{p}) \in X\ \wedge\ (\overrightarrow{d'}) = op(\overrightarrow{d})\ \wedge\ \overrightarrow{d}' \neq \perp\ \wedge\ \overrightarrow{p}' = \overrightarrow{p}\}$.

Cette fonction permet d'obtenir tous les successeurs des éléments contenus dans X et n'en calcule pas plus. Si à partir de l'ensemble X, l'application de la transition ne donne que des contenus de mémoire vides ($\perp$), l'ensemble X' sera alors l'ensemble vide. Aussi, dans l'algorithme, savoir si la transition est applicable c'est tester si le résultat du $post_t$ est vide ou non.

3.5.2.3 *Post** effectivement constructible

Comme nous en avons discuté au début de cette partie, itérer infiniment un circuit à partir d'une configuration peut donner un ensemble infini de configurations. Il est alors impossible de construire cet ensemble entièrement. Pour ce faire, il nous faut *accélérer* la construction, c'est-à-dire pouvoir calculer en un pas l'ensemble potentiellement infini des configurations atteignables par l'itération d'un circuit θ un nombre non borné de fois. Soit

γ, γ' deux configurations d'un automate $\mathcal{A}$, l'itération infini d'un circuit θ se note de la manière suivante : $\gamma \xrightarrow{\theta^*} \gamma'$.

Dans chacun des cas que nous analysons dans les chapitres suivants nous donnons des algorithmes (souvent partiels) caractérisant l'itération des circuits, nous verrons qu'ils appartiennent à différentes catégories (décrites ci-dessous).

Propriété 3 ($Post^*$ effectivement constructible)

Soit $\mathcal{A} = (Q,\ \mathcal{X},\ \mathcal{P},\ L,\ Op,\ T)$ un automate étendu. Soit $\theta = t_1 t_2 \cdots t_l$ un circuit de $\mathcal{A}$. Nous distinguons 3 sortes d'algorithme post* permettant d'accélerer :

- *post* sur-approximation : cet algorithme calcule au moins toutes les configurations atteignables mais peut donner un ensemble qui en contient plus.*

$$\forall X_1 \in E^{\mathbb{D}}\ \forall M_1, M_2 \in Rep\ \forall X_2 \in E^{\mathbb{D}}.$$

$$M_1 = rep(X_1)\ \wedge\ M_2 = post^*(\theta, M_1)\ \wedge\ X_2 = ens(M_2) \Rightarrow$$
$$\forall(\overrightarrow{d}, \overrightarrow{p}) \in X_1\ \exists(\overrightarrow{d}', \overrightarrow{p}') \in X_2.\ (q, \overrightarrow{d}, \overrightarrow{p}) \xrightarrow{\theta^*} (q, \overrightarrow{d}', \overrightarrow{p}')$$

- *post* sous-approximation : le résultat de cet algorithme donne seulement des configurations atteignables mais ne les contient pas forcément toutes.*

$$\forall X_1 \in E^{\mathbb{D}}\ \forall M_1, M_2 \in Rep\ \forall X_2 \in E^{\mathbb{D}}.$$

$$M_1 = rep(X_1)\ \wedge\ M_2 = post^*(\theta, M_1)\ \wedge\ X_2 = ens(M_2) \Rightarrow$$
$$\forall(\overrightarrow{d}', \overrightarrow{p}') \in X_2\ \exists(\overrightarrow{d}, \overrightarrow{p}) \in X_1.(q, \overrightarrow{d}, \overrightarrow{p}) \xrightarrow{\theta^*} (q, \overrightarrow{d}', \overrightarrow{p}')$$

- *post* exact : le résultat de l'algorithme contient tout l'ensemble des configurations atteignables et seulement celui-là. C'est la conjonction des deux conditions ci-dessus.*

3.5.2.4 Construction en profondeur d'abord

Nous avons défini dans les paragraphes ci-dessus les bases de notre calcul des configurations atteignables. En premier lieu, nous avons utilisé une représentation symbolique, nous étendons ci-après la définition des configurations (cf. paragraphe 2.3) aux configurations symboliques qui tiennent comptent des ensembles de valuations pour les variables.

Définition 25 (Configuration symbolique)

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op, T)$ un automate étendu. Une configuration symbolique $\gamma = (q, X)$ est l'association d'un état de contrôle $q \in Q$ et d'un ensemble de valuations des variables et des paramètres $X \in E^{\mathbb{D}}$.

Nous pouvons donc maintenant redéfinir la notion de sémantique pour un automate étendu que nous appelons *graphe symbolique*.

Définition 26 (Graphe symbolique)

La sémantique "symbolique" d'un automate étendu $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P},\ L, Op, T)$ est le système de transitions étiqueté (ou graphe symbolique) $\mathcal{S}(\mathcal{A}) = (C, L, R^e)$ tel que :

- *$C = Q \times E^{\mathbb{D}}$ est l'ensemble des configurations symboliques de l'automate.*
- *$R^e \subseteq C \times L \times C$ est la* relation d'atteignabilité en un pas symbolique *de $\mathcal{S}$.*

La relation R^e tient compte du calcul de l'ensemble des configurations atteignables par accélération. En effet, un triplet $((q_1,\ X_1), l, (q_2,\ X_2))$ appartient à R^e, ce qui s'écrit $(q_1,\ X_1) \xrightarrow{l}_{R^e} (q_2,\ X_2)$ si :

- soit il existe une transition $(q,\ l,\ op,\ q_2) \in T$ telle que : $X_2 = \{(\overrightarrow{\mathrm{d}}', \overrightarrow{\mathrm{p}}') \mid (\overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}) \in X_1 \ \wedge\ \overrightarrow{\mathrm{d}}' = op(\overrightarrow{\mathrm{d}}) \ \wedge\ d' \neq \perp \ \wedge\ \overrightarrow{\mathrm{p}}' = \overrightarrow{\mathrm{p}}\}$.
- soit il existe un algorithme d'accélération $post^*$ et il existe un circuit θ sur l'état de contrôle q_1 tel que l'étiquette associée à ce circuit est l, on a alors les conditions suivantes :

$$q_2 = q_1 \ \wedge\ \exists M_1, M_2 \in Rep. M_1 = rep(X_1) \ \wedge\ M_2 = post^*(\theta, M_1) \ \wedge\ X_2 = ens(M_2)$$

Nous pouvons maintenant calculer l'ensemble des configurations symboliques atteignables d'un automate étendu. Supposons que la représentation symbolique des ensembles de valuations des variables et des paramètres soit une bonne représentation et que les algorithmes $post_t$ et $post^*$ existent effectivement pour cette représentation, le principe de l'algorithme de calcul des configurations dans le cas symbolique est alors le même que celui exposé dans la section des systèmes finis (section 3.5.1). Il suffit de commencer le calcul avec un ensemble ne contenant que la configuration symbolique initiale. Puis d'appliquer itérativement la relation d'atteignabilité en un pas R^e de l'automate étendu en ajoutant à l'ensemble des configurations atteignables toutes les configurations obtenues par l'application de la relation. En fait, pour être réellement capables d'écrire cet algorithme il nous manque le moyen de savoir si nous avons un circuit ou non. Pour cela, nous ajoutons à l'algorithme "énumératif" la sauvegarde du chemin (listes des transitions par lesquelles on est arrivé à une configuration donnée) à chaque appel à la fonction *explore*. Ainsi, il suffit de remonter cette liste jusqu'à trouver une configuration ayant le même état de contrôle pour trouver les circuits. Nous supposons que cette recherche de circuit est effectuée par la fonction *donner_circuit* qui renvoie l'ensemble des circuits par rapport à un état de contrôle donné.

L'algorithme de calcul en profondeur symbolique est représenté par la figure 3.4. L'algorithme travaille sur la représentation des ensembles de valuations des variables.

fonction EnProfondeurSymb (AMSP $(\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op, T))$) :
ensemble de configurations symboliques.
lexique :
états_visités : ensemble de configurations symboliques.
chemin : liste de transitions.

procédure explore_symb (configuration $\gamma = (q, M)$, liste de conf. *chemin*)
algo. :
états_visités := *états_visités* $\cup \{(q, M)\}$

pour tous circuits $\theta \in donner\_circuit(q, chemin)$ **faire**
soit $M_2 = post^*(\theta, M)$ **dans**
si ($(M_2 \neq \emptyset)$ **et puis** $((q', M_2) \notin$ *états_visités*$)$)
alors explore_symb((q', M_2))

pour toutes transitions $t = (q, l, op, q') \in T$ **faire**
soit $M_2 = post_t(M)$ **dans**
si $((M_2 \neq \emptyset)$ **et puis** $((q', M_2) \notin$ *états_visités*$)$)
alors *chemin* := Ajout_fin(*chemin*, t)
explore_symb $((q', M_2),$ *chemin*)
finalgo.

algo. :
états_visités $= \emptyset$
chemin := liste vide
explore_symb$((q_0, rep(\{\vec{d}_0, \vec{p}_0\})),$ *chemin*)
retourner *états_visités*
finalgo.

FIG. 3.4 – Exploration symbolique en profondeur de l'espace des configurations

3.6 Composition de modèles

Supposons maintenant que le domaine de définition des variables soit un produit $\mathbb{D} = \mathbb{D}_1 \times \cdots \times \mathbb{D}_n$.

Pour analyser un tel système nous distinguons deux techniques.

La première est de trouver une représentation symbolique générale représentant des éléments de $\mathbb{D}$. Il suffit alors d'appliquer le principe que nous venons de décrire. C'est à dire montrer qu'il est possible d'appliquer une transition sur un ensemble de valuations de $\mathbb{D}$, de faire des opérations telles que l'inclusion, l'intersection entre deux représentations d'ensemble et de pouvoir (si besoin est) appliquer un algorithme d'accélération sur cette représentation.

Pour la seconde technique nous supposons qu'il est trop difficile de concevoir une

représentation générale. Supposons alors que pour chacun des domaines de définition pris séparément il existe une représentation sur laquelle il est possible d'appliquer une transition, d'appliquer les opérations de bases et éventuellement d'accélérer.

Observons quelques cas particuliers de manipulation de domaines différents avant de donner un principe général quant à l'analyse d'automates étendus dont le domaine des variables est un produit.

Commençons par observer un automate contenant des compteurs et des horloges représenté par la figure 3.5. Cet automate contient deux horloges h_1 et h_2 et un compteur c. Il a deux états de contrôle.

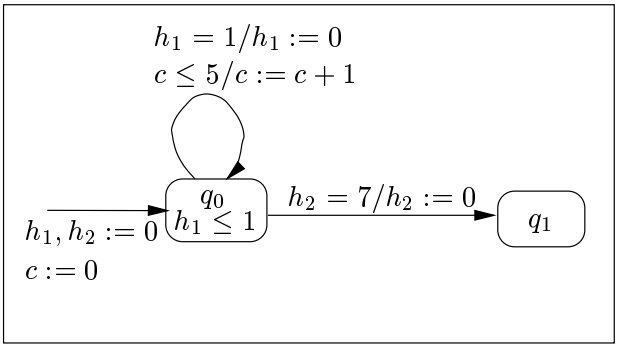


FIG. 3.5 – Combinaison : un exemple d'accélération

La question qui se pose alors est de savoir si l'état q_1 est atteignable.

La configuration initiale de cet exemple est :

$$\gamma_0 = (q_0,\ 0 \leq h_1, h_2 \leq 1\ \wedge\ h_1 = h_2,\ c = 0)$$

La boucle peut s'exécuter et nous accélérons donc chacun des domaines. Nous obtenons alors pour les horloges l'ensemble de valuations suivant :

$$0 \leq h_1 \leq 1\ \wedge\ 1 + n_0 \leq h_2 \leq 2 + n_0\ \wedge\ h_2 - h_1 = 1 + n_0\ \wedge\ n_0 \geq 0$$

Pour les compteurs nous obtenons :

$$c = 1 + n_1\ \wedge\ 0 \leq n_1 \leq 5$$

Avec ces valeurs, la garde de la transition menant à l'état q_1 est satisfaisable en posant par exemple $n_0 = 6$. Pourtant, l'état q_1 ne devrait pas être atteignable car il existe un lien entre les compteurs et les horloges. En effet, puisque c'est la même transition qu'empruntent les horloges et le compteur nous devrions avoir $n_0 = n_1$ comme condition supplémentaire. Si cette condition n'est pas ajoutée aux ensembles de contraintes alors nous calculons une sur-approximation de l'ensemble des configurations atteignables.

Observons maintenant un automate composé de compteurs et de files d'attente avec perte (figure 3.6). La sémantique des files d'attente avec perte implique que le nombre

d'exécutions d'une boucle n'est pas conservé par la représentation. Aussi, dans l'exemple, l'état q_2 est atteignable : intuitivement le compteur c_2 peut prendre n'importe quelle valeur puisque le contenu de la file dans l'état q_1 est a^*. Pourtant, la file ne devrait contenir, au plus, que 10 fois le message a. Et donc le compteur c_2 ne devrait pas dépasser la valeur de 10. Nous obtenons dans cet exemple une sur-approximation de l'ensemble des configurations atteignables car nous utilisons la file d'attente pour compter.

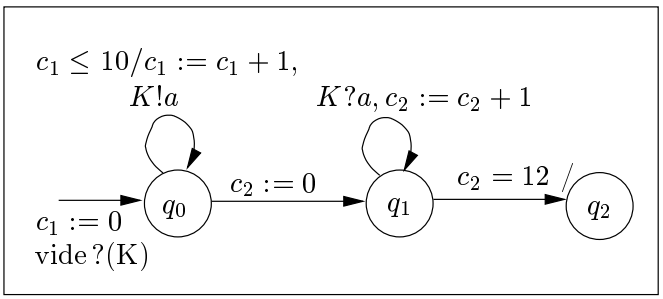


FIG. 3.6 – Combinaison compteurs et files d'attente

Dans le cadre général des automates étendus composés, nous allons classer les ensembles de variables en deux catégories :

- *Les variables qui comptent* : c'est-à-dire celles pour lesquelles la représentation choisie permet de garder une trace du nombre d'itérations d'un circuit effectuées. Nous notons cet ensemble $\mathcal{C}$.
- *Les variables qui ne comptent pas* : c'est-à-dire celles pour lesquelles la représentation choisie ne conserve pas de lien avec le nombre d'itérations des circuits. Nous notons cet ensemble $\tilde{\mathcal{C}}$.

La configuration symbolique est alors $\gamma = (q, M_1, M_2, \cdots, M_{k_x}, lien)$ où M_i représente un ensemble de valuations des variables de l'ensemble $\mathcal{X}_i$ et *lien* est une représentation du lien entre les différentes représentations. Le lien doit satisfaire les propriétés suivantes :

Propriété 4

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op, T)$ un automate étendu complexe. Pour tout $i \in [1, k_x]$, soit $M_i \in Rep_i$ un objet de la représentation d'un ensemble de valuations des variables contenues dans l'ensemble $\mathcal{X}_i$. La représentation du lien et les représentations choisies pour les variables qui comptent doivent avoir les propriétés suivantes :

- *Il existe une représentation du lien vide. Nous le notons $\emptyset$.*
- *Pour toutes les représentations des variables l'algorithme d'application d'une transition doit être effectivement constructible c'est-à-dire*

$$\forall M_i \exists M_i'. Post(M_i, lien) = (M_i', lien)$$

- *La représentation du lien doit pouvoir créer un nouveau lien lors d'une accélération sur les variables qui comptent. Nous notons cette fonction creer_lien.*

Chapitre 4

Analyse des systèmes à compteurs et horloges paramétrés

Nous donnons dans ce chapitre la représentation que nous avons choisie pour représenter des ensembles de valuations de compteurs et/ou d'horloges. Cette représentation est une extension des matrices à différence de bornes utilisées dans le cadre de l'analyse des systèmes à horloges (sans paramètres). Nous avons choisi de pouvoir représenter dans un même objet les valuations des compteurs et des horloges. Nous avons vu dans le chapitre 3 qu'il nous fallait pouvoir garder un lien entre les différentes variables de différents domaines de définition. C'est encore plus vrai si l'on considère des protocoles contenant le mécanisme suivant. Supposons un protocole faisant une certaine tâche jusqu'à expiration d'un time-out. Ce time-out ayant expiré, le protocole décide alors de recommencer la tâche en augmentant le temps imparti. Une façon de représenter cette manière de fonctionner est d'utiliser un compteur pour garder la valeur du time-out. En effet, un paramètre ne suffit plus, car dans nos modèles nous ne pouvons changer sa valeur. La figure 4.1 représente un tel protocole.

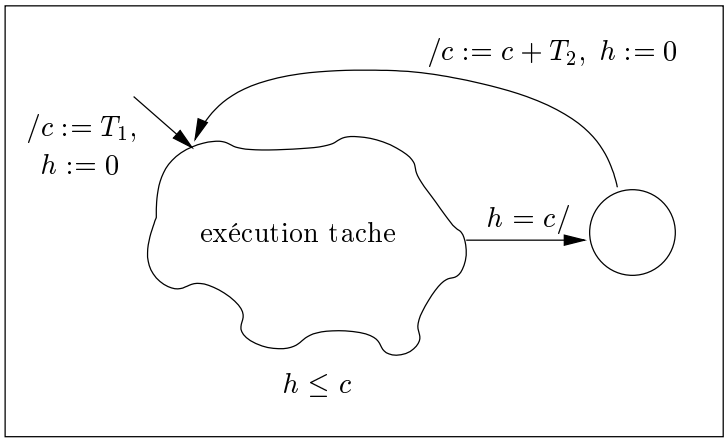


FIG. 4.1 – Un exemple de composition

Dans cet exemple, il nous faudra donc conserver la différence de valeur entre l'horloge et le compteur pour pouvoir tester la garde $h = c$.

Ce chapitre commence par la définition formelle des matrices de bornes paramétrées puis nous montrons qu'il est possible d'appliquer toute transition sur ces matrices. La section suivante est consacrée à la notion d'accélération, nous montrons alors effectivement comment nous calculons en un pas l'ensemble des configurations atteignables d'un circuit.

La réalisation de toutes ces opérations (inclusion de matrices, application d'une transition, ...) va, dans la plupart des cas, être réduite à la satisfaction d'une formule d'arithmétique. Nous discutons dans la section 4.4 des problèmes de décision sur les formules.

Nous terminons ce chapitre par quelques exemples d'analyse de systèmes à compteurs, à horloges ou hétérogènes.

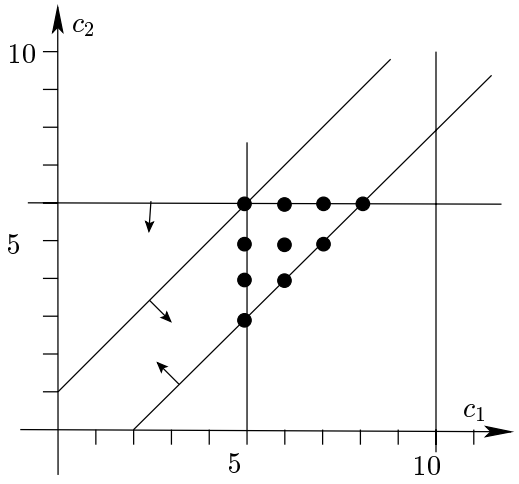
4.1 Matrice de bornes paramétrées (PDBM : Parametric Difference Bounded Matrix)

Nous donnons ici, les opérations possibles sur ces nouvelles matrices de façon syntaxique. Ainsi, souvent nous allons nous ramener au test d'une formule arithmétique. Nous discutons de la décidabilité des formules dans la section 4.4 (page 73).

4.1.1 Les matrices de bornes

Une matrice de bornes est une manière de représenter des contraintes arithmétiques. Cette structure a été inventée par Measche et Berthomieu [MB83] et Dill [Dil89]. Elle est utilisée notamment pour la représentation symbolique du contenu des horloges dans les systèmes temporisés [Yov93, Yov98].

Par exemple, l'ensemble des valuations des compteurs c_1, c_2 exprimé par la contrainte $\phi = 5 \leq c_1 \leq 10 \ \wedge \ 0 \leq c_2 \leq 6 \ \wedge \ 1 \leq c_1 - c_2 \leq 2$ est représenté graphiquement par le dessin suivant (fig. 4.2 (a)) ou par la matrice de borne M_1.



(a) Représentation graphique

$$M_1 = \begin{pmatrix} & c_0 & c_1 & c_2 \\ c_0 & (\leq,0) & (\leq,-5) & (\leq,0) \\ c_1 & (\leq,10) & (\leq,0) & (\leq,2) \\ c_2 & (\leq,6) & (\leq,-1) & (\leq,0) \end{pmatrix}$$

(b) Matrice de bornes

FIG. 4.2 – Représentations de ϕ

Nous remarquons que la matrice contient 3 compteurs. En effet, chaque contrainte $c_i \leq t$ est remplacée par la contrainte $c_i - c_0 \leq t$ où c_0 est un nouveau compteur tel que sa valeur est toujours 0. Chaque entrée de la matrice représente la borne supérieure de la différence entre 2 compteurs. Par exemple, l'entrée $M_{c_1c_2}$ est égale à $(\leq, 2)$ ce qui représente la contrainte $c_1 - c_2 \leq 2$. Les entrées $M_{c_1c_0}$ et $M_{c_0c_2}$ représentent respectivement les bornes supérieures et inférieures du compteur c_1. Nous remarquons que nous utilisons pour cela le compteur fictif c_0. Les éléments de la diagonale sont toujours égaux à $(\leq, 0)$.

Nous ne détaillons pas plus avant dans ce travail les opérations sur les matrices de bornes non paramétrées, pour plus de renseignements le lecteur peut se référer à la thèse de Sergio Yovine [Yov93].

4.1.2 Extension des matrices de bornes : matrices de bornes paramétrées

Nous commençons par définir la notion de bornes puis nous définissons les matrices à différences de bornes paramétrées.

4.1.2.1 Les bornes paramétrées

Une borne est l'association d'un symbole $\{<, \leq\}$ et d'un terme sur les paramètres. L'ensemble des bornes, noté $\mathcal{B}$, est défini par :

$$\mathcal{B} = \{<, \leq\} \times AT(\mathcal{P}) \ \cup \ \{(<, -\infty), (<, \infty)\}$$

Puisque les bornes mettent en œuvre les paramètres, nous définissons la notion de *bornes contraintes*. Une borne contrainte est l'association d'une borne paramétrée et d'une formule contraignant les paramètres. L'ensemble des bornes paramétrées contraintes, noté $\tilde{\mathcal{B}}$ est défini de la manière suivante :

$$\tilde{\mathcal{B}} = \mathcal{B} \times F(\mathcal{P})$$

En fonction de la nature des paramètres (réels, entiers) les formules sont considérées comme formules sur les entiers ou sur les réels. Nous discutons de ce point dans la section 4.4.

Les symboles de relation $<$ et $\leq$ sont totalement ordonnés : $<$ est supposé strictement plus petit que $\leq$. Nous pouvons définir une relation d'ordre $\sqsubseteq$ sur les bornes paramétrées contraintes de la manière suivante :

Soient $((\prec_1, t_1), \phi_1), ((\prec_2, t_2), \phi_2) \in \tilde{\mathcal{B}}$ deux bornes paramétrées contraintes, nous avons alors $((\prec_1, t_1), \phi_1) \sqsubseteq ((\prec_2, t_2), \phi_2)$ si et seulement si la formule ϕ_{inc} est satisfaisable.

$$\phi_{inc} = \forall p_i \in \mathcal{P} \ \phi_1 \Rightarrow \phi_2 \ \wedge \ ((t_1 < t_2) \ \vee \ (t_1 = t_2 \ \wedge \ \prec_1 \leq \prec_2))$$

Notons que la borne $(<, \infty)$ satisfait pour tout $\tilde{b} \in \tilde{\mathcal{B}}, \tilde{b} \sqsubseteq ((<, \infty), vrai)$.

Nous définissons **l'ordre strict** $\sqsubset$ entre deux bornes paramétrées contraintes de la manière suivante : $((\prec_1, t_1), \phi_1) \sqsubset ((\prec_2, t_2), \phi_2)$ si et seulement si la formule ϕ_{incs} est

satisfaisable.

$$\phi_{incs} = \forall p_i \in \mathcal{P}\ \phi_1 \Rightarrow \phi_2\ \wedge\ (\ (t_1 < t_2)\ \vee\ (t_1 = t_2 \wedge\ \prec_1 < \prec_2)\ )$$

Nous notons $\tilde{b}_1 = \tilde{b}_1$ si et seulement si il est possible d'avoir à la fois $\tilde{b}_1 \sqsubseteq \tilde{b}_2$ et $\tilde{b}_1 \sqsupseteq \tilde{b}_2$.

Nous introduisons maintenant deux opérations $\oplus$ et $\otimes$. Ces opérations vont nous être utiles à la caractérisation d'opérations sur les matrices telles que intersection ou canonisation. Intuitivement, à partir des contraintes $x - y < 5$ et $y - z \leq 3$, nous pouvons déduire $x - z < 8$, c'est ce que propose de calculer l'opération $\otimes$. Pour calculer l'intersection de deux contraintes nous devons être capable de déterminer de deux bornes laquelle est la plus petite, c'est ce que calcule l'opération $\oplus$.

Définition 27 ($\oplus$, $\otimes$)
Soient $\tilde{b}_1 = ((\prec_1, t_1), \phi_1)$ et $\tilde{b}_2 = ((\prec_2, t_2), \phi_2)$ deux bornes paramétrées contraintes. Nous définissons alors les opérations suivantes :

- $\otimes\ :\ \tilde{\mathcal{B}} \times \tilde{\mathcal{B}} \longrightarrow \tilde{\mathcal{B}}$ *tel que*

$$\tilde{b}_1 \otimes \tilde{b}_2 = ((min(\prec_1, \prec_2), t_1 + t_2), \phi_1\ \wedge\ \phi_2)$$

où pour tout $t \in AT(\mathcal{P})$ nous avons :

$$\begin{array}{rcl} t + \infty & = & \infty \\ t + (-\infty) & = & -\infty \\ \infty + \infty & = & \infty \\ \infty + (-\infty) & = & \infty \\ (-\infty) + (-\infty) & = & -\infty \end{array}$$

Nous notons que :

* *la borne contrainte $((\leq, 0), vrai)$ est* l'élément neutre *pour l'opération $\otimes$: en effet, $((\leq, 0), vrai)$ satisfait pour tout $\tilde{b} \in \tilde{\mathcal{B}}$, $((\leq, 0), vrai) \otimes \tilde{b} = \tilde{b}$.*
* *la borne contrainte $((<, \infty), faux)$ est* l'élément absorbant *pour l'opération $\otimes$: en effet, $((<, \infty), faux)$ satisfait pour tout $\tilde{b} \in \tilde{\mathcal{B}}$, $((<, \infty), faux) \otimes \tilde{b} = ((<, \infty), faux)$.*

- *Pour définir l'opération $\oplus$, nous avons besoin de déterminer le minimum entre deux termes. Nous posons tout d'abord les trois formules suivantes :*

$$\begin{array}{rcl} \Phi_1 & \equiv & \exists p \in \mathcal{P}.\ \phi_1\ \wedge\ \phi_2\ \wedge\ t_1 < t_2 \\ \Phi_2 & \equiv & \exists p \in \mathcal{P}.\ \phi_1\ \wedge\ \phi_2\ \wedge\ t_1 = t_2 \\ \Phi_3 & \equiv & \exists p \in \mathcal{P}.\ \phi_1\ \wedge\ \phi_2\ \wedge\ t_1 > t_2 \end{array}$$

Définissons maintenant $\oplus\ :\ \tilde{\mathcal{B}} \times \tilde{\mathcal{B}} \longrightarrow 2^{\tilde{\mathcal{B}}}$ tel que

$$\tilde{b}_1 \oplus \tilde{b}_2 = min(\tilde{b}_1, \tilde{b}_2)$$

$$= \begin{cases} \textit{1. si } \Phi_1 \wedge \neg\Phi_2 \wedge \neg\Phi_3 \\ \quad \{ ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_1 < t_2) \} \\ \textit{2. si } \neg\Phi_1 \wedge \Phi_2 \wedge \neg\Phi_3 \wedge \prec_1 \leq \prec_2 \\ \quad \{ ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_2 = t_1) \} \\ \textit{3. si } \neg\Phi_1 \wedge \Phi_2 \wedge \neg\Phi_3 \wedge \prec_1 \geq \prec_2 \\ \quad \{ ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_2 = t_1) \} \\ \textit{4. si } \neg\Phi_1 \wedge \neg\Phi_2 \wedge \Phi_3 \\ \quad \{ ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_2 < t_1) \} \\ \textit{5. si } \Phi_1 \wedge \Phi_2 \wedge \neg\Phi_3 \wedge \prec_1 \leq \prec_2 \\ \quad \{ ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_2 \leq t_1) \} \\ \textit{6. si } \Phi_1 \wedge \Phi_2 \wedge \neg\Phi_3 \wedge \prec_1 \geq \prec_2 \\ \quad \{ ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_1 < t_2),\ ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_1 = t_2) \} \\ \textit{7. si } \Phi_1 \wedge \neg\Phi_2 \wedge \Phi_3 \\ \quad \{ ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_2 < t_1),\ ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_1 < t_2) \} \\ \textit{8. si } \neg\Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \prec_2 \leq \prec_1 \\ \quad \{ ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_2 \leq t_1) \} \\ \textit{9. si } \neg\Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \prec_2 \geq \prec_1 \\ \quad \{ ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_2 = t_1),\ ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_1 > t_2) \} \\ \textit{10. si } \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \prec_1 \leq \prec_2 \\ \quad \{ ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_1 < t_2),\ ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_1 = t_2), \\ \quad\quad ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_1 > t_2) \} \\ \textit{11. si } \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \prec_1 \geq \prec_2 \\ \quad \{ ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_2 < t_1),\ ((\prec_2, t_2), \phi_1 \wedge \phi_2 \wedge t_2 = t_1), \\ \quad\quad ((\prec_1, t_1), \phi_1 \wedge \phi_2 \wedge t_2 > t_1) \} \end{cases}$$

Nous notons que :

* *la borne contrainte* $((<, \infty), vrai)$ *est* l'élément neutre *pour l'opération* $\oplus$ *: en effet,* $((<, \infty), vrai)$ *satisfait pour tout* $\tilde{b} \in \tilde{\mathcal{B}}$, $((<, \infty), vrai) \oplus \tilde{b} = \{\tilde{b}\}$.

Soient $\tilde{b}_1, \tilde{b}_2, \tilde{b}_3 \in \tilde{\mathcal{B}}$ 3 bornes paramétrées contraintes, nous avons alors :

$$(\tilde{b}_1 \oplus \tilde{b}_2) \oplus \tilde{b}_3 = \{\tilde{b}_i \mid \forall \tilde{b}' \in \tilde{b}_1 \oplus \tilde{b}_2,\ \tilde{b}_i \in \tilde{b}' \oplus \tilde{b}_3\}$$

4.1.2.2 Les matrices de bornes paramétrées contraintes

Pour pouvoir représenter une contrainte de la forme $\wedge\, x_i - x_j \leq t_{ij}$ (où $x_i, x_j \in \mathcal{X}$ peuvent être des compteurs et/ou des horloges) par une matrice, nous introduisons une variable fictive x_0 dont la valeur est toujours 0. Nous pouvons ainsi exprimer la condition $x_i \leq t_i$ par la contrainte $x_i - x_0 \leq t_i$. Nous avons maintenant $\mathcal{X} = \mathcal{X} \cup \{x_0\}$.

Définition 28 (Matrices de bornes paramétrées)

Soit $\mathcal{X}$ *un ensemble de* $n_x + 1$ *variables (contenant une variable fictive* x_0*). Toute contrainte sur les variables est définie par une conjonction de contraintes de la forme*

$x_i - x_j \prec t$ où $\prec \in \{<, \leq\}$, $x_i, x_j \in \mathcal{X}$ et $t \in AT(\mathcal{P})$ peut être représentée par une matrice carrée M de dimension $n_x + 1$ telle que :

- *Les éléments de la matrice appartiennent à $\mathcal{B}$.*
- *La première colonne de la matrice encode les bornes supérieures des variables. Si $x_i - x_0 \prec t$ apparaît dans la contrainte, alors $M_{i0} = (\prec, t)$, sinon $M_{i0} = (<, \infty)$: la variable x_i est dite non bornée.*
- *La première ligne de la matrice encode les bornes inférieures des variables. Si $x_0 - x_i \prec t$ apparait dans la contrainte, alors $M_{0i} = (\prec, t)$, sinon $M_{0i} = (\leq, 0)$ la variable (que ce soit compteur ou horloge) x_i prenant ses valeurs respectivement dans $\mathbb{N}$ ou $\mathbb{R}^+$.*
- *L'élément M_{ij} de la matrice pour tout $i, j > 0$ est une paire $(\prec, t)$ qui encode la contrainte $x_i - x_j \prec t$. Si aucune contrainte entre x_i et x_j n'apparaît dans la conjonction, l'élément M_{ij} est égal à $(<, \infty)$.*

Soit M une matrice à bornes paramétrées, nous écrirons plutôt la formule $x - y \prec M_{xy}$ à la place de la formule $x - y \prec t$ pour représenter la contrainte associée à l'élément $M_{xy} = (\prec, t)$.

Comme dans le cadre des bornes paramétrées, nous définissons la notion de matrice paramétrée contrainte. Une *matrice de bornes paramétrées contrainte* $\tilde{M} = (M, \phi)$ est l'association d'une matrice de bornes paramétrées M et d'une contrainte $\phi \in F(\mathcal{P})$.

Soit $\tilde{M} = (M, \phi)$ une matrice de bornes paramétrées contrainte, *l'ensemble caractéristique* de $\tilde{M}$, noté $[\![\tilde{M}]\!]$ est l'ensemble des valuations des compteurs, des horloges et des paramètres de l'automate $(\overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$ qui satisfont la formule exprimée par la matrice contrainte telles que $\overrightarrow{\nu} = (\overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}) \in [\![\wedge_{i=0}^{n_x} \wedge_{j=0}^{n_x} x_i - x_j \prec_{ij} t_{ij} \ \wedge \ \phi]\!]$ pour toute variable $x_i, x_j \in \mathcal{X}$ où $M_{x_i x_j} = (\prec_{ij}, t_{ij})$. Soit un ensemble de matrices contraintes $\{\tilde{M}_1, \cdots, \tilde{M}_n\}$, nous définissons l'ensemble caractéristique de cet ensemble de matrices de la manière suivante : $[\![\{\tilde{M}_1, \cdots, \tilde{M}_n\}]\!] = \cup_{i \in [1 \cdots n]} [\![\tilde{M}_i]\!]$.

Avant d'étendre les opérations $\oplus$ et $\otimes$, nous définissons une matrice particulière $\hat{M}$ telle que ces éléments sont des ensembles de bornes paramétrées contraintes. Nous notons un élément de $\hat{M}$ par $\hat{M}_{ij} = \{\tilde{b}_{ij}^1, \cdots, \tilde{b}_{ij}^{n_{ij}}\}$ où $n_{ij} = |\hat{M}_{ij}|$ est le nombre de bornes contraintes de cet ensemble. A partir de $\hat{M}$, nous donnons l'ensemble des matrices de bornes paramétrées contraintes qu'elle contient par la fonction *mat* suivante. L'ensemble contient alors $\Pi_{i=0}^{n_c} \Pi_{j=0}^{n_c} |\hat{M}_{ij}|$ matrices paramétrées contraintes.

$$mat\left(\begin{pmatrix} \{\tilde{b}_{00}^1 \cdots \tilde{b}_{00}^{n_{00}}\} & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \{\tilde{b}_{nn}^1 \cdots \tilde{b}_{nn}^{n_{nn}}\} \end{pmatrix}\right) =$$

$$\{ \begin{array}{l} \left(\begin{pmatrix} b^1_{00} & \cdots & b^1_{0n} \\ \cdots & \cdots & \cdots \\ b^1_{n0} & \cdots & b^1_{nn} \end{pmatrix}, \ \phi^1_{00} \wedge \phi^1_{01} \wedge \cdots \wedge \phi^1_{nn} \right) \ ; \\ \left(\begin{pmatrix} b^2_{00} & \cdots & b^1_{0n} \\ \cdots & \cdots & \cdots \\ b^1_{n0} & \cdots & b^1_{nn} \end{pmatrix}, \ \phi^2_{00} \wedge \phi^1_{01} \wedge \cdots \wedge \phi^1_{nn} \right) \ ; \\ \cdots \\ \left(\begin{pmatrix} b^{n_{00}}_{00} & \cdots & b^{n_{0n}}_{0n} \\ \cdots & \cdots & \cdots \\ b^{n_{n0}}_{n0} & \cdots & b^{n_{nn}}_{nn} \end{pmatrix}, \ \phi^{n_{00}}_{00} \wedge \phi^{n_{01}}_{01} \wedge \cdots \wedge \phi^{n_{nn}}_{nn} \right) \ ; \end{array} \}$$

Les opérations $\oplus$ et $\otimes$ peuvent être étendues aux matrices de bornes paramétrées contraintes de la façon suivante :

- Soient $\tilde{M}_1 = (M_1, \phi_1)$ et $\tilde{M}_2 = (M_2, \phi_2)$ deux matrices de bornes paramétrées contraintes. $\tilde{M}_1 \otimes \tilde{M}_2$ est l'ensemble de matrices $mat(\hat{M})$, la matrice $\hat{M}$ étant définie telle que pour toute variable $x_i, x_j \in \mathcal{X}$,

$$\hat{M}_{ij} = \bigoplus_{x_k \in \mathcal{X}} (\tilde{M}_{1,x_i x_k}, \phi_1) \otimes (\tilde{M}_{2,x_k x_j}, \phi_2)$$

- Soient $\tilde{M}_1 = (M_1, \phi_1)$ et $\tilde{M}_2 = (M_2, \phi_2)$ deux matrices de bornes paramétrées contraintes. $\tilde{M}_1 \oplus \tilde{M}_2$ est l'ensemble de matrices $mat(\hat{M})$, la matrice $\hat{M}$ étant définie telle que pour toute variable $x_i, x_j \in \mathcal{X}$,

$$\hat{M}_{ij} = (\tilde{M}_{1,x_i x_j}, \phi_1) \oplus (\tilde{M}_{2,x_i x_j}, \phi_2)$$

Soit $\tilde{M} = (M, \phi)$ une matrice de bornes paramétrées contrainte.
Soit $\{((\prec_1, t_1), \phi_1), \cdots, ((\prec_n, t_n), \phi_n)\} = (M_{xy}, \phi) \otimes (M_{yz}, \phi)$ tel que $x, y, z \in \mathcal{X}$. Nous écrirons $x - z \prec (M_{xy}, \phi) \otimes (M_{yz}, \phi)$ à la place de la formule $\bigvee_{i=1}^{n} x - y \prec_i t_i \ \wedge \ \phi_i$.

Remarque : Soient $\tilde{M}_1 = (M_1, \phi_1)$ et $\tilde{M}_2 = (M_2, \phi_2)$ deux matrices de bornes paramétrées contraintes. Soit $\{(M'_1, \phi'_1), \cdots, (M'_n, \phi'_n)\} = \tilde{M}_1 \ op \ \tilde{M}_2$ où $op \in \{\oplus, \otimes\}$, les formules ϕ'_i sont des restrictions de la formules $\phi_1 \wedge \phi_2$. Ces formules satisfont la propriété suivante

$$\forall i \in [1, n] \ \phi'_i \Rightarrow \phi_1 \ \wedge \ \phi_2 \tag{4.1}$$

Nous définissons les matrices $\tilde{E} = (E, vrai)$ et $\tilde{N} = (N, faux)$ de la façon suivante. Pour toutes variables $x_1, x_2 \in \mathcal{X}$, nous avons :

$$\begin{array}{lcl} E_{x_1 x_2} & = & \begin{cases} (\leq, 0) & \text{si } x_1 = x_2 \\ (<, \infty) & \text{sinon} \end{cases} \\ N_{x_1 x_2} & = & (<, \infty) \end{array}$$

Notons de plus que :

- $\tilde{E}$ est l'élément neutre pour $\otimes$, i.e. $\tilde{E} \otimes \tilde{M} = \{\tilde{M}\}$ pour toute matrice contrainte $\tilde{M}$.

– $\tilde{N}$ est l'élément absorbant pour $\otimes$, i.e. $\tilde{N} \otimes \tilde{M} = \{\tilde{M}\}$ pour toute matrice contrainte $\tilde{M}$.

Soit $\tilde{\mathcal{M}}$ l'ensemble des matrices de bornes paramétrées contraintes définit par :

$$\tilde{\mathcal{M}} = \{\tilde{M} = (M, \phi) \mid \forall x \in \mathcal{X}. \tilde{M}_{xx} = (\leq, 0)\} \cup \tilde{N}$$

Soit $\tilde{M} \in \tilde{\mathcal{M}}$ une matrice de bornes paramétrées contrainte. Nous définissons l'opération "$*$" de la manière suivante :

$$\tilde{M}^* = \bigoplus_{k \in \mathbb{N}} \tilde{M}^k$$

où $\tilde{M}^0 = \tilde{E}$ et $\tilde{M}^{k+1} = \tilde{M}^k \otimes \tilde{M}$.

Remarque : Soit $\tilde{M} \in \tilde{\mathcal{M}}$, $\tilde{M} \neq \tilde{N}$, telle que $[\![\tilde{M}]\!] \neq \emptyset$. Nous avons $((\leq, 0), vrai) \sqsubseteq \tilde{M}_{xx}$ pour tout $x \in \mathcal{X}$. En effet, si nous supposions le contraire, puisque $[\![\tilde{M}]\!] \neq \emptyset$, il existe au moins une valuation $\overrightarrow{\nu} \in [\![\tilde{M}]\!]$ telle que $(x - x < 0)[\mathcal{X} \cup \mathcal{P} / \overrightarrow{\nu}]$ doit être valide, ce qui est une contradiction. Nous avons donc bien, par conséquent $M_{xx} = (\leq, 0)$.

Nous voyons dans la propriété suivante que les matrices contraintes contenues dans les ensembles résultats des opérations définies ci-dessus sont distinctes deux à deux, ce qui implique qu'une valuation des variables donnée n'appartient qu'à une seule matrice.

Propriété 5

Soient $\tilde{M}$ et $\tilde{M}'$ deux matrices de bornes paramétrées contraintes et $\{\tilde{M}^1, \cdots, \tilde{M}^n\} = \tilde{M}$ op $\tilde{M}'$ où op $\in \{\oplus, \otimes\}$, $\tilde{M}^i = (M^i, \phi^i)$ pour tout $i \in [1, n]$. Alors les matrices $\tilde{M}^i$ sont distinctes deux-à-deux, ou encore la formule suivante n'est pas satisfaisable :

$$\exists\, p \in \mathcal{P} \bigvee_{\substack{i,j \in [1,n] \\ i \neq j}} \phi^i \wedge \phi^j$$

Nous avons alors aussi la propriété suivante :

$$\forall i \in [1, n]\ \forall \overrightarrow{v} \in [\![\tilde{M}^i]\!]\ \ \nexists j \in [1, n].\ j \neq i \Rightarrow \overrightarrow{v} \in [\![\tilde{M}^j]\!]$$

4.1.3 Inclusion entre deux matrices de bornes paramétrées contraintes

L'odre total $\sqsubseteq$ défini sur les bornes paramétrées contraintes peut être étendu de manière naturelle aux matrices de bornes paramétrées contraintes. En effet, pour toute paire de matrices paramétrées contraintes $\tilde{M} = (M, \phi)$ et $\tilde{M}' = (M', \phi')$, nous avons :

$$\tilde{M} \sqsubseteq \tilde{M}' \text{ si } \forall x_i, x_j \in \mathcal{X}\ (M_{x_i x_j}, \phi) \sqsubseteq (M'_{x_i x_j}, \phi')$$

Nous avons alors la propriété suivante :

$$\text{si } \tilde{M} \sqsubseteq \tilde{M}' \text{ alors } [\![\tilde{M}]\!] \subseteq [\![\tilde{M}']\!] \tag{4.2}$$

Deux matrices de bornes paramétrées contraintes sont égales $\tilde{M} = \tilde{M}'$ si et seulement si $\tilde{M} \sqsubseteq \tilde{M}'$ et $\tilde{M} \sqsupseteq \tilde{M}'$.

Nous pouvons étendre la définition de l'inclusion et de l'égalité à des ensembles de matrices. Soient $\tilde{\mathcal{M}}_1 = \{\tilde{M}_1, \cdots, \tilde{M}_n\}$ et $\tilde{\mathcal{M}}_2 = \{\tilde{M}'_1, \cdots, \tilde{M}'_m\}$ deux ensembles de matrices de bornes paramétrées contraintes, $\{\tilde{M}_1, \cdots, \tilde{M}_n\} \sqsubseteq \{\tilde{M}'_1, \cdots, \tilde{M}'_m\}$ si et seulement si $\forall i \in [1,n]\ \exists j \in [1,m]\ .\ \tilde{M}_i \sqsubseteq \tilde{M}'_j$. Nous dirons que $\tilde{\mathcal{M}}_1 = \tilde{\mathcal{M}}_2$ si et seulement si $\tilde{\mathcal{M}}_1 \sqsubseteq \tilde{\mathcal{M}}_2$ et $\tilde{\mathcal{M}}_1 \sqsupseteq \tilde{\mathcal{M}}_2$.

Remarque : Le test d'inclusion (défini ci-dessus) entre deux ensembles de matrices n'est pas complet. En effet, puisque l'union de deux matrices contraintes n'est pas, en général, une matrice contrainte (l'union de deux ensembles convexes n'est pas forcément un ensemble convexe), nous n'utilisons pas l'opération union. Il se peut alors qu'une matrice $\tilde{M}_i$ soit couverte par plusieurs matrices $\tilde{M}'_j$. Le test tel que nous l'avons écrit ci-dessus n'est alors pas capable de trouver cette inclusion. Notre test est un test "sûr" au sens où lorsque la réponse est vraie (l'inclusion est vérifiée), l'ensemble caractéristique de $\tilde{\mathcal{M}}_1$ est bien compris dans celui de $\tilde{\mathcal{M}}_2$. Si le test répond faux, nous ne pouvons alors rien dire au sujet de l'inclusion.

4.1.4 Intersection entre les matrices de bornes paramétrées

Prenons tout d'abord un exemple. Soient $\mathcal{X} = \{x, y, z\}$ trois variables et $\mathcal{P} = \{T_1, T_2, T_3, T_4\}$ quatre paramètres. Soient les formules ψ_1 et ψ_2 suivantes :

$$\begin{array}{rcl} \psi_1 & = & T_1 \leq x \ \wedge\ z \leq T_2 \ \wedge\ x - y \leq 3 \ \wedge\ T_1 \geq 0 \ \wedge\ T_2 \geq 0 \\ \psi_2 & = & x \leq T_3 \ \wedge\ y \leq 5 \ \wedge\ z \leq T_4 \ \wedge\ y - z < 1 \ \wedge\ T_3 \geq 0 \ \wedge\ T_4 \geq 0 \end{array}$$

Les matrices de bornes paramétrées contraintes $\tilde{M}_1 = (M_1, \phi_1)$ et $\tilde{M}_2 = (M_2, \phi_2)$ ci-dessous correspondent respectivement à ψ_1 et ψ_2. Calculons alors l'ensemble de matrice $\tilde{M}_1 \oplus \tilde{M}_2$:

$$M_1 = \begin{array}{c|cccc} & x_0 & x & y & z \\ \hline x_0 & (\leq,0) & (\leq,-T_1) & (\leq,0) & (\leq,0) \\ x & (<,\infty) & (\leq,0) & (\leq,3) & (<,\infty) \\ y & (<,\infty) & (<,\infty) & (\leq,0) & (<,\infty) \\ z & (\leq,T_2) & (<,\infty) & (<,\infty) & (\leq,0) \end{array} \quad \oplus \quad M_2 = \begin{array}{c|cccc} & x_0 & x & y & z \\ \hline x_0 & (\leq,0) & (\leq,0) & (\leq,0) & (\leq,0) \\ x & (\leq,T_3) & (\leq,0) & (<,\infty) & (<,\infty) \\ y & (\leq,5) & (<,\infty) & (\leq,0) & (<,1) \\ z & (\leq,T_4) & (<,\infty) & (<,\infty) & (\leq,0) \end{array}$$

$$\phi_1 = T_1 \geq 0 \ \wedge\ T_2 \geq 0 \qquad\qquad \phi_2 = T_3 \geq 0 \ \wedge\ T_4 \geq 0$$

donne

$$\left\{ \begin{array}{l} \left(\begin{array}{c|cccc} & x_0 & x & y & z \\ \hline x_0 & (\leq,0) & (\leq,-T_1) & (\leq,0) & (\leq,0) \\ x & (\leq,T_3) & (\leq,0) & (\leq,3) & (<,\infty) \\ y & (\leq,5) & (<,\infty) & (\leq,0) & (<,1) \\ z & (\leq,\mathbf{T_2}) & (<,\infty) & (<,\infty) & (\leq,0) \end{array} \right) \quad T_1 \geq 0 \ \wedge \ T_2 \geq 0 \ \wedge \ T_3 \geq 0 \ \wedge \ \mathbf{T_2} < \mathbf{T_4} \\ \\ \left(\begin{array}{c|cccc} & x_0 & x & y & z \\ \hline x_0 & (\leq,0) & (\leq,-T_1) & (\leq,0) & (\leq,0) \\ x & (\leq,T_3) & (\leq,0) & (\leq,3) & (<,\infty) \\ y & (\leq,5) & (<,\infty) & (\leq,0) & (<,1) \\ z & (\leq,\mathbf{T_4}) & (<,\infty) & (<,\infty) & (\leq,0) \end{array} \right) \quad T_3 \geq 0 \ \wedge \ T_4 \geq 0 \ \wedge \ T_1 \geq 0 \ \wedge \ \mathbf{T_2} > \mathbf{T_4} \\ \\ \left(\begin{array}{c|cccc} & x_0 & x & y & z \\ \hline x_0 & (\leq,0) & (\leq,-T_1) & (\leq,0) & (\leq,0) \\ x & (\leq,T_3) & (\leq,0) & (\leq,3) & (<,\infty) \\ y & (\leq,5) & (<,\infty) & (\leq,0) & (<,1) \\ z & (\leq,\mathbf{T_4}) & (<,\infty) & (<,\infty) & (\leq,0) \end{array} \right) \quad T_1 \geq 0 \ \wedge \ T_2 \geq 0 \ \wedge \ T_3 \geq 0 \ \wedge \ \mathbf{T_2} = \mathbf{T_4} \end{array} \right\}$$

ce qui correspond à la contrainte $\psi_1 \ \wedge \ \psi_2$.

En effet, comme l'illustre l'exemple, l'opération $\oplus$ sur les matrices coïncide avec l'intersection des ensembles caractéristiques.

Lemme 1

Soient $\tilde{M}$ et $\tilde{M}'$ deux matrices de bornes paramétrées contraintes, nous avons alors :

$$[\![\tilde{M} \oplus \tilde{M}']\!] = [\![\tilde{M}]\!] \cap [\![\tilde{M}']\!]$$

Preuve.

Soient $\tilde{M} = (M, \phi)$ et $\tilde{M}'(M', \phi')$ deux matrices de bornes paramétrées contraintes. Soit $\tilde{M}''$ une matrice appartenant à l'ensemble $\tilde{M} \oplus \tilde{M}'$.

- $\subseteq$ Par définition de l'opération $\oplus$, nous avons $\tilde{M}'' \sqsubseteq \tilde{M}$ et $\tilde{M}'' \sqsubseteq \tilde{M}'$. Nous avons donc $[\![\tilde{M}'']\!] \subseteq [\![\tilde{M}]\!]$ et $[\![\tilde{M}'']\!] \subseteq [\![\tilde{M}']\!]$. Par conséquent, $[\![\tilde{M}'']\!] \subseteq [\![\tilde{M}]\!] \cap [\![\tilde{M}']\!]$.
  Puisque $\tilde{M}''$ est une matrice prise au hasard dans l'ensemble des matrices résultats, la démonstration ci-dessus s'applique pour toutes les matrices du résultat de $\oplus$, nous avons donc $[\![\tilde{M} \oplus \tilde{M}']\!] \subseteq [\![\tilde{M}]\!] \cap [\![\tilde{M}']\!]$.
- $\supseteq$ Soit $(\overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}) \in [\![\tilde{M}]\!] \cap [\![\tilde{M}']\!]$. Nous avons pour toutes variables $x, y \in \mathcal{X}$ les formules suivantes qui sont satisfaites :
  $(x - y \prec \tilde{M}_{xy} \ \wedge \ \phi)[\mathcal{X}/\overrightarrow{\mathrm{d}}, \mathcal{P}/\overrightarrow{\mathrm{p}}]$
  $(x - y \prec \tilde{M}'_{xy} \ \wedge \ \phi')[\mathcal{X}/\overrightarrow{\mathrm{d}}, \mathcal{P}/\overrightarrow{\mathrm{p}}]$
  Nous pouvons donc dire qu'il existe une borne contrainte $\tilde{b} = (b, \phi'') \in (\tilde{M}_{xy} \oplus \tilde{M}'_{xy})$ telle que la formule $(x - y \prec b \ \wedge \ \phi'')[\mathcal{X}/\overrightarrow{\mathrm{d}}, \mathcal{P}/\overrightarrow{\mathrm{p}}]$ est satisfaisable. Par conséquent il existe une matrice $\tilde{M}'' \in \tilde{M} \oplus \tilde{M}'$ par définition de $\oplus$ telle que $d \in [\![\tilde{M}'']\!]$.

■

4.1.5 Forme normale d'une matrice de bornes paramétrées

Nous allons voir que plusieurs matrices paramétrées contraintes peuvent représenter le même ensemble caractéristique. Nous voyons qu'il est alors possible de ne considérer qu'une seule de ces matrices (une seule matrice parmi celles représentant le même ensemble caractéristique) que nous appelons matrice canonique. Il est aussi possible à partir de n'importe quelle matrice contrainte de donner la (ou les) matrice(s) canonique(s) qui lui est (sont) associée(s). En effet à cause des contraintes sur les paramètres il n'existe pas une seule matrice canonique mais un ensemble tel que la conjonction des contraintes de chaque matrice prise 2 à 2 n'est pas satisfaisable.

Prenons tout d'abord un exemple. Soit $\mathcal{X} = \{x, y\}$ l'ensemble des variables que nous considérons. Soit $\mathcal{P} = \{T_1\}$ l'ensemble des paramètres qui apparaissent dans notre exemple. Soit la formule $\psi = T_1 \leq x \leq 5 \wedge 2 \leq y \leq 5 \wedge x - y \leq 5 \wedge y - x \leq 3 \wedge T_1 \geq 0$, la matrice de bornes paramétrées contrainte $\tilde{M} = (M, \phi)$ la représentant est la suivante :

$$M = \begin{pmatrix} & 0 & x & y \\ 0 & (\leq,0) & (\leq,-T_1) & (\leq,-2) \\ x & (\leq,5) & (\leq,0) & (\leq,5) \\ y & (\leq,5) & (\leq,3) & (\leq,0) \end{pmatrix} \quad \phi = \quad 0 \leq T1 \leq 5$$

Nous pouvons appliquer l'opération $\otimes$ entre 2 exemplaires de cette matrice $\tilde{M} \otimes \tilde{M}$ ce qui nous donne l'ensemble de matrices suivant :

$$\left\{ \begin{array}{cc} \begin{pmatrix} & 0 & x & y \\ 0 & (\leq,0) & (\leq,-T_1) & (\leq,-2) \\ x & (\leq,5) & (\leq,0) & (\leq,3) \\ y & (\leq,5) & (\leq,5-T_1) & (\leq,0) \end{pmatrix} & \begin{pmatrix} & 0 & x & y \\ 0 & (\leq,0) & (\leq,-T_1) & (\leq,-2) \\ x & (\leq,5) & (\leq,0) & (\leq,3) \\ y & (\leq,5) & (\leq,3) & (\leq,0) \end{pmatrix} \\ 2 \leq T_1 \leq 4 & 0 \leq T_1 \leq 1 \\ \begin{pmatrix} & 0 & x & y \\ 0 & (\leq,0) & (\leq,-T_1) & (\leq,-2) \\ x & (\leq,5) & (\leq,0) & (\leq,3) \\ y & (\leq,5) & (\leq,5-T_1) & (\leq,0) \end{pmatrix} & \begin{pmatrix} & 0 & x & y \\ 0 & (\leq,0) & (\leq,-T_1) & (\leq,-2) \\ x & (\leq,5) & (\leq,0) & (\leq,3) \\ y & (\leq,5) & (\leq,3) & (\leq,0) \end{pmatrix} \\ T_1 = 5 & T_1 = 2 \end{array} \right\}$$

L'ensemble caractéristique représenté par la matrice $\tilde{M}$ et par l'ensemble de matrice résultat de l'opération $\tilde{M} \otimes \tilde{M}$ est le même ensemble de valuations. Avant d'exprimer formellement ce résultat, regardons dans l'exemple que la formule ψ contient des *contraintes implicites* qui ont été révélées à l'aide de l'opération $\otimes$.

En effet,

$$\underbrace{x \leq 5}_{\phi_1} \wedge \underbrace{-y \leq -2}_{\phi_2} \Rightarrow \underbrace{x - y \leq 3}_{\phi_3}$$

La contrainte contenue initialement dans ψ : $x - y \leq 5$ était donc trop lache car elle peut être remplacée par la contrainte ϕ_3. Observons maintenant un détail de

l'opération $\otimes$. La contrainte ϕ_1 correspond à la borne $M_{x0} = (\leq, 5)$ dans la matrice M et la borne $M_{0y} = (\leq, -2)$ correspond à la contrainte ϕ_2. Pour trouver l'élément M'_{xy} des matrices résultats, d'après la définition de l'opération $\otimes$ il nous faut calculer $(M_{x0} \otimes M_{0y}) \oplus (M_{xx} \otimes M_{xy}) \oplus (M_{xy} \otimes M_{yy})$. Nous ne tiendrons pas compte des paramètres dans ce calcul (car les bornes mises en jeu ne contiennent pas de paramètres). Nous avons les calculs suivants :

$$\begin{array}{lcl} M_{x0} \otimes M_{oy} & = & (\leq, 3) \\ M_{xx} \otimes M_{xy} & = & (\leq, 5) \\ M_{xy} \otimes M_{yy} & = & (\leq, 5) \end{array} \tag{4.3}$$

Ainsi, $M'_{xy} = (\leq, 3)$, l'opération $\otimes$ appliquée à la matrice $\tilde{M}$ revient à réduire les contraintes implicites.

Nous remarquons que l'ensemble des valuations représenté par la matrice contrainte initiale est le même que l'ensemble représenté par l'union des ensembles caractéristiques de chacune des matrices $\tilde{M'}$ contenues dans l'ensemble résultat de $\tilde{M} \otimes \tilde{M}$. Chaque matrice $\tilde{M'}$ est alors telle que $\tilde{M'} \otimes \tilde{M'} = \{\tilde{M'}\}$.

Toute matrice contrainte peut être mise sous forme canonique. Il nous faut tout d'abord choisir une matrice pour représenter l'ensemble vide. Pour cela, nous choisissons la matrice $\tilde{O} = (O, \phi_o)$ telle que pour toutes variables $c_1, c_2 \in \mathcal{X}$ nous ayons :

$$O_{c_1 c_2} = (<, -\infty) \ \wedge \ \phi_o = vrai$$

Nous définissons alors la fonction cf qui à partir de toute matrice de bornes paramétrées contrainte $\tilde{M} \in \mathcal{M}$ donne l'ensemble des matrices canoniques.

$$cf(\tilde{M}) = \begin{cases} \{\tilde{O}\} & \text{si } [\![\tilde{M}]\!] = \emptyset \\ \tilde{M}^* & \text{sinon} \end{cases}$$

Les matrices canoniques sont telles que chaque élément $\tilde{M}'_{xy}$ de ces matrices contienne les plus petites valeurs exprimant la distance entre les variables x et y appartenant à $\mathcal{X}$. Nous ne pouvons trouver une troisième variable z tel que passer par z donne une valeur plus petite, ce qui s'exprime par la propriété suivante : Soient $\tilde{M}$ une matrice de bornes paramétrées contrainte, $\{\tilde{M}_1, \cdots, \tilde{M}_n\} = cf(\tilde{M})$ l'ensemble de ses matrices canoniques, nous avons alors :

$$\forall \tilde{M}_i = (M', \phi') \in cf(\tilde{M}) \ \forall x, y, z \in \mathcal{X} \ (M'_{xy}, \phi') \sqsubseteq (M'_{xz}, \phi') \otimes (M'_{zy}, \phi') \tag{4.4}$$

Nous montrons maintenant que $cf(\tilde{M})$ représente bien l'ensemble des matrices canoniques de $\tilde{M}$. Pour cela montrons tout d'abord que l'ensemble caractéristique de $\tilde{M}$ est égal à l'ensemble caractéristique de $\tilde{M} \otimes \tilde{M}$. Ainsi (par corollaire) nous aurons $[\![\tilde{M}^k]\!] = [\![\tilde{M}]\!]$ et enfin nous prouvons que $[\![\tilde{M}]\!] = [\![cf(\tilde{M})]\!]$.

Lemme 2

Pour toute matrice de bornes paramétrées contrainte $\tilde{M} \in \mathcal{M}$,

$$si \ [\![\tilde{M}]\!] \neq \emptyset \ alors \ [\![\tilde{M} \otimes \tilde{M}]\!] = [\![\tilde{M}]\!].$$

La preuve de ce lemme est donnée en annexe page 159. Du lemme 2 découle le corollaire suivant.

Corollaire 2

Pour toute matrice de bornes paramétrées contrainte $\tilde{M} \in \mathcal{M}$,

$$si\ [\![\tilde{M}]\!] \neq \emptyset,\ alors\ pour\ tout\ k \geq 1, [\![M^k]\!] = [\![M]\!].$$

Montrons maintenant que les matrices $\tilde{M}$ et $cf(\tilde{M})$ représentent le même ensemble caractéristique.

Lemme 3

Pour toute matrice de bornes paramétrées contrainte $\tilde{M} \in \mathcal{M}$,

$$cf(\tilde{M})\ satisfait\ [\![\tilde{M}]\!] = [\![cf(\tilde{M})]\!]$$

Preuve. Si $[\![\tilde{M}]\!] = \emptyset$ alors $cf(\tilde{M}) = \{\tilde{O}\}$. Comme $\tilde{O}$ satisfait $[\![\tilde{O}]\!] = \emptyset$, nous avons bien $[\![\tilde{M}]\!] = [\![\{\tilde{O}\}]\!]$.

Si $[\![\tilde{M}]\!] \neq \emptyset$ alors $cf(M) = M^*$. D'après le lemme 1, $[\![\oplus_{k\in\mathbb{N}} \tilde{M}^k]\!] = \cup_{k\in\mathbb{N}} [\![\tilde{M}^k]\!]$. Le corollaire 2 nous permet de dire que $[\![\tilde{M}^k]\!] = [\![\tilde{M}]\!]$ pour tout $k \geq 1$. Ainsi, $[\![cf(\tilde{M})]\!] = [\![\tilde{E}]\!] \cup [\![M]\!]$ (d'après la définition de *), nous avons donc $[\![cf(\tilde{M})]\!] = [\![M]\!]$.

■

Montrons maintenant que pour toute matrice de bornes paramétrées contrainte l'ensemble des matrices obtenues par $cf(\tilde{M})$ est l'ensemble le plus petit permettant de représenter l'ensemble caractéristique de $\tilde{M}$.

Lemme 4

Pour toutes matrices de bornes paramétrées contraintes $\tilde{M}, \tilde{M}' \in \mathcal{M}$.
Soit $\{\tilde{M}^1, \cdots, \tilde{M}^n\} = cf(\tilde{M})$ *l'ensemble des matrices canoniques de* $\tilde{M}$ *alors*

$$[\![\tilde{M}]\!] = [\![\tilde{M}']\!] \neq \emptyset \ \Rightarrow \forall j \in [1, n]\ \tilde{M}^j \sqsubseteq \tilde{M}'$$

La preuve de ce lemme est donnée en annexe page 159. Montrons alors que toutes les matrices équivalentes (ayant le même ensemble caractéristique) ont la même forme canonique.

Proposition 4

Pour toutes matrices de bornes paramétrées contraintes $\tilde{M}$, $\tilde{M}' \in \mathcal{M}$, *nous avons*

$$[\![\tilde{M}]\!] = [\![\tilde{M}']\!]\ ssi\ cf(M') = cf(\tilde{M}')$$

Preuve. Soient $\tilde{M}$ et $\tilde{M}'$ deux matrices de bornes paramétrées contraintes.

$\Leftarrow$ si $cf(\tilde{M}) = cf(\tilde{M}')$ alors d'après le lemme 3 nous avons bien $[\![\tilde{M}]\!] = [\![\tilde{M}']\!]$.

$\Rightarrow$ Si $[\![\tilde{M}]\!] = [\![\tilde{M}']\!] = \emptyset$ alors $cf(\tilde{M}) = cf(\tilde{M}') = \{\tilde{O}\}$ par définition de cf.
Si $[\![\tilde{M}]\!] = [\![\tilde{M}']\!] \neq \emptyset$ alors d'après le lemme 3 nous avons $[\![cf(\tilde{M})]\!] = [\![cf(\tilde{M}')]\!]$. Or, $cf(\tilde{M})^* = cf(\tilde{M})$. Donc par le lemme 4, $cf(\tilde{M}) \sqsubseteq cf(\tilde{M}')$ et par symétrie $cf(\tilde{M}') \sqsubseteq cf(\tilde{M})$.

■

Pour toute matrice $\tilde{M}$, l'ensemble de matrices $cf(\tilde{M})$ peut être calculé par l'algorithme de Floyd-Warshall [AHU74] étendu aux ensembles de matrices. Ceci nous permet aussi de déterminer si l'ensemble caractéristique de $\tilde{M}$ est vide. En effet, si chacune des matrices obtenues par l'application de l'algorithme de Floyd-Warshall est telle qu'un élément de la diagonale est inférieur à $(\leq, 0)$ l'ensemble caractéristique de chacune des matrices est alors vide et donc celui de $\tilde{M}$ est vide aussi.

4.2 Application d'une transition — $post_t$

Nous venons de définir la représentation des contenus des compteurs, des horloges et des paramètres, caractérisons maintenant l'algorithme permettant d'appliquer une transition. Une configuration symbolique est de la forme $\gamma = (q, \tilde{M})$ où q est un état de contrôle de l'automate étendu et $\tilde{M} \in \tilde{\mathcal{M}}$ est une matrice de bornes paramétrées contrainte représentant les valuations.

Tout d'abord, regardons comment transformer les opérations sur les vecteurs de compteurs ou d'horloges en opérations sur les matrices de bornes contraintes.

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, I, Op_g, Op_v, T)$ un système à compteurs et à horloges paramétrés. Chaque opération $o_v \in Op_v$ peut se représenter par un couple $(A, \overrightarrow{\mathrm{b}})$ où :

- A est une matrice carrée de dimension $(n_x+1)\times(n_x+1)$, chaque ligne de cette matrice ne contient qu'un seul 1. La matrice est de dimension $n_x + 1$ car nous transformons les opérations de la forme $x_i := t_i$ en $x_i := x_0 + t_i$ en utilisant la variable fictive x_0 introduite pour les matrices. La matrice A appartient à l'ensemble Mat^A défini de la façon suivante :

$$\forall A \in Mat^A.$$
$$(\forall\, i \in [0, n_x]\ \exists j \in [0, n_x]\ .\ A[i,j] = 1\ \wedge\ \forall\, k \in [0, n_x]\ k \neq j \Rightarrow A[i,k] = 0)$$

- $\overrightarrow{\mathrm{b}}$ est un vecteur de dimension $n_x + 1$ tel que :

$$\overrightarrow{\mathrm{b}}(0) = 0\ \wedge\ \forall i \in [1, n_x]\ \exists t_i \in ATl(\mathcal{P})\ \overrightarrow{\mathrm{b}}(i) = t_i$$

  Nous remarquons que pour les horloges $t_i = 0$.

Une transition t est représentée par le n-upplet suivant : (q_1, l, o_g, o_v, q_2) où q_1 et q_2 sont des états de contrôle, o_g la garde de la transition, $o_v = (A, \overrightarrow{\mathrm{b}})$ les affectations des compteurs et des horloges.

Une garde est une conjonction de contraintes de la forme $x_i - x_j \prec t$ où $x_i, x_j \in \mathcal{X}$ et $t \in AT(\mathcal{P})$. Toute garde o_g peut donc être représentée par une matrice de bornes paramétrées contrainte $\tilde{M}_g \in \tilde{\mathcal{M}}$. Savoir si un ensemble de valuations des variables du système satisfait la garde revient alors à effectuer l'intersection entre la matrice représentant la garde et la matrice représentant les valuations des variables. Si la canonisation du résultat de l'intersection est l'ensemble $\{\tilde{O}\}$, alors le système ne peut pas prendre cette transition (aucune valuation ne satisfaisait cette garde).

Nous venons de voir qu'une opération d'affectation pour les compteurs et les horloges est un couple $(A, \overrightarrow{\mathrm{b}})$.

Pour manipuler des matrices de bornes paramétrées, nous transformons les vecteurs $\overrightarrow{\mathrm{b}}$ en matrices carrées B de dimension $(n_x+1)\times(n_x+1)$ de la façon suivante :

$$\forall x_i, x_j \in \mathcal{X}, B[x_i,x_j] = \overrightarrow{\mathrm{b}}(x_i) - \overrightarrow{\mathrm{b}}(x_j)$$

La matrice ainsi construite appartient à l'ensemble Mat^B défini par :

$$Mat^B = \{\ B \mid \forall x_i,x_j \in \mathcal{X}\ B[x_i,x_j] \in AT(\mathcal{P})\ \wedge\ B[x_i,x_j] = -B[x_j,x_i] \wedge$$

$$\forall x_k \in \mathcal{X}.\ B[x_i,x_k] + B[x_k,x_j] = B[x_i,x_j]\ \}$$

Soit $(Mat^A)^T$ l'ensemble des matrices transposées de Mat^A (elles ont donc un seul 1 par colonne). Avant d'exprimer l'effet d'une transition sur une matrice contrainte nous allons surcharger les opérations $*$ et $+$ de la manière suivante :

- L'opération $* : Mat^A \times \tilde{\mathcal{M}} \longrightarrow \tilde{\mathcal{M}}$ est telle que

$$\forall A \in Mat^A\ \forall \tilde{M} = (M,\phi) \in \tilde{\mathcal{M}}\ \exists \tilde{M}' = (M',\phi') = A * \tilde{M} \in \tilde{\mathcal{M}}.$$

$$\forall i,j \in [0,n_x]\ M'_{ij} = M_{i'j} \text{ où } A[i,i'] = 1\ \wedge\ \phi' = \phi$$

- L'opération $* : \tilde{\mathcal{M}} \times (Mat^A)^T \longrightarrow \tilde{\mathcal{M}}$ est telle que

$$\forall A \in Mat^A\ \forall \tilde{M} = (M,\phi) \in \tilde{\mathcal{M}}\ \exists \tilde{M}' = (M',\phi') = \tilde{M} * A^T \in \tilde{\mathcal{M}}.$$

$$\forall i,j \in [0,n_c]\ M'_{ij} = M_{ij'} \text{ où } A^T[j',j] = 1\ \wedge\ \phi' = \phi$$

- L'opération $+ : \tilde{\mathcal{M}} \times Mat^B \longrightarrow \tilde{\mathcal{M}}$ est telle que

$$\forall B \in Mat^B\ \forall \tilde{M} = (M,\phi) \in \tilde{\mathcal{M}}\ \exists \tilde{M}' = (M',\phi') = \tilde{M} + B \in \tilde{\mathcal{M}}.$$

$$\forall i,j \in [0,n_c]\ M'_{ij} = (\prec_{ij}, t_{ij} + B[ij]) \text{ où } M_{ij} = (\prec_{ij}, t_{ij})\ \wedge\ \phi' = \phi$$

Les ensembles Mat^A et Mat^B possèdent quelques bonnes propriétés que nous détaillons ci-dessous :

Propriété 6

Soient $A_1, A_2 \in Mat^A$, $B_1, B_2 \in Mat^B$, nous avons alors :

1. *$(Mat^A, *, I)$ est un monoïde ($*$ est la multiplication standard sur les matrices, I est la matrice identité).*
2. *$(Mat^B, +, 0)$ est un monoïde ($+$ est l'opération d'addition standard sur les matrices et 0 est la matrice nulle).*
3. *Mat^A est fermé par la multiplication de matrices contenues dans Mat^A, i.e.*

$$\forall A_1, A_2 \in Mat^A\ \exists A_3 \in Mat^A.\ A_1 * A_2 = A_3$$

où $*$ est la multiplication standard sur les matrices.

4. *Mat^B est fermé par addition, i.e*

$$\forall B_1, B_2 \in Mat^B\ \exists B_3 \in Mat^B.\ B_1 + B_2 = B_3$$

où $+$ est l'addition standard sur les matrices.

5. *Mat^B est fermé par multiplication avec une matrice appartenant à Mat^A et sa transposé, i.e.*

$$\forall A_1 \in Mat^A \, \forall B_1 \in Mat^B \, \exists B_2 \in Mat^B. \; A_1 * B_1 * A_1^T = B_2$$

où $*$ est la multiplication standard entre matrice.

6. *Soient $\overrightarrow{p}$ une valuation sur les paramètres $\mathcal{P}$ et $B \in Mat^B$, nous avons alors $|[\![B]\!]_{\overrightarrow{p}}| = 1$, i.e. la matrice B représente un point pour une valuation des paramètres donnée.*

La preuve de cette propriété est donnée en annexe à la page 161.

Le sixième item de la propriété implique que pour chacune des matrices $B \in Mat^B$ il existe un unique vecteur $\overrightarrow{\mathrm{b}}$ de dimensions $n_c + 1$ correspondant au point décrit par B. Soit $\overrightarrow{\mathrm{b}}$ un vecteur et $B' \in Mat^B$ une matrice, nous utilisons alors les notations suivantes :

- La fonction *matrice* associe à chaque vecteur la matrice le représentant : $matrice(\overrightarrow{\mathrm{b}}) = B$.
- La fonction *vecteur* est l'opération inverse, elle permet de retrouver le vecteur associé à une matrice : $vecteur(B') = \overrightarrow{\mathrm{b}}'$.

Ainsi, appliquer la transition $x := Ax + \overrightarrow{\mathrm{b}}$ sur un ensemble de configurations décrit par une matrice de bornes paramétrées contrainte $\tilde{M}$ est la matrice contrainte $\tilde{M}' = A * \tilde{M} * A^T + B$ où A^T est la matrice transposée de A.

L'effet d'une transition sur les configurations symboliques est donc représenté par un couple de matrices $(A, B) \in Mat^A \times Mat^B$.

Il nous faut maintenant caractériser le passage du temps pour les horloges. Si nous nous souvenons que la première colonne d'une matrice représente les bornes supérieures de chacune des horloges, nous concevons sans peine que laisser passer le temps revient à mettre la borne $(<, \infty)$ à la place de toutes les bornes supérieures des horloges. Nous définissons l'opération $\uparrow$ sur les matrices contraintes de la façon suivante. Soit $(\tilde{M}) = (M, \phi)$ une matrice contrainte, la matrice $\uparrow \tilde{M} = (M', \phi)$ est telle que ses éléments sont les suivants :

$$M'_{ij} = \begin{cases} (<, \infty) & \text{si } i \neq 0 \ \wedge \ j = 0 \ \wedge \ x_i \text{ est une horloge} \\ M_{ij} & \text{sinon} \end{cases}$$

Soit $t = (q, l, \tilde{M}_g, (A, B), q')$ une transition du système. Soit $\gamma = (q, \tilde{M})$ une configuration symbolique de ce système. Nous pouvons maintenant définir effectivement la notion de $post_t$ sur une configuration symbolique de la manière suivante :

$$post_t((q, \tilde{M})) = \{(q', \tilde{M}'') \mid \forall \tilde{M}' \in \tilde{M} \otimes \tilde{M}_g. \; \tilde{M}' \neq \tilde{O} \Rightarrow \tilde{M}'' = \uparrow \ (A * \tilde{M}' * A^T + B)\}$$

Ainsi, si aucune valuation des variables contenue dans le graphe symbolique ne satisfaisait la garde de la transition, l'ensemble donné par $post_t$ est alors vide.

4.3 Effet d'un circuit — $post^*$

Dans cette section, nous voyons tout d'abord qu'il nous faut étendre la définition des matrices de bornes paramétrées en permettant la manipulation de variables représentant

le nombre d'itération d'un circuit. Nous explicitons ensuite notre algorithme permettant de calculer en un pas l'ensemble des configurations atteignables d'un circuit, puis nous discutons de l'exactitude de l'ensemble des valuations représenté par les configurations atteintes.

4.3.1 Matrices de bornes paramétrées ouvertes

Pour représenter le nombre d'exécution d'un circuit nous introduisons des variables particulières appartenant à l'ensemble $\mathcal{N}$, nommées *variables d'itération.*

Nous appelons *borne paramétrée fermée* les bornes que nous avons définies dans la section 4.1.2.1 (page 53), nous appelons *borne paramétrée ouverte* les bornes que nous définissons ci-dessous pouvant manipuler des variables appartenant à $\mathcal{N}$. Une borne paramétrée ouverte est l'association d'un symbole $\{<, \leq\}$ et d'un terme sur les paramètres et les variables d'itération. L'ensemble des bornes ouverte, noté $\mathcal{B}^{\mathcal{O}}$, est défini par :

$$\mathcal{B}^{\mathcal{O}} = \{<, \leq\} \times AT(\mathcal{P} \cup \mathcal{N}) \cup \{(<, \infty), (<, -\infty)\}$$

Une *borne paramétrée ouverte contrainte* $\tilde{b}^o = (b^o, \phi)$ est l'association d'une borne paramétrée ouverte $b^o \in \mathcal{B}^{\mathcal{O}}$ et d'une contrainte sur les paramètres $\phi \in F(\mathcal{P} \cup \mathcal{N})$.

Les variables d'itération apparaissant dans deux bornes ouvertes ne sont pas forcément les mêmes. Nous notons par $\phi(p, n)$ le fait que dans la formule ϕ apparaîssent des paramètres nommés p et des variables d'itération n.

Nous allons redéfinir la notion d'ordre entre deux bornes paramétrées contraintes ouvertes. Soient $((\prec_1, t_1), \phi_1)$ et $((\prec_2, t_2), \phi_2)$ deux bornes ouvertes, $((\prec_1, t_1), \phi_1) \sqsubseteq ((\prec_2, t_2), \phi_2)$ si et seulement si la formule ϕ_{inc} suivante est satisfaisable.

$$\phi_{inc} = \forall p_i \in \mathcal{P}, \forall n \in \mathcal{N}.\ \phi_1(p, n) \Rightarrow$$

$$\exists m \in \mathcal{N}.\ (\phi_2(p, m)\ \wedge\ (t1(p, n) < t_2(p, m)\ \vee\ (t1(p, n) = t_2(p, m) \wedge\ \prec_1 \leq \prec_2)))$$

L'ordre strict sur les bornes ouvertes est défini de la manière suivante :

$$\phi_{incs} = \forall p_i \in \mathcal{P}, \forall n \in \mathcal{N}\ \phi_1(p, n) \Rightarrow$$

$$\exists m \in \mathcal{N}\ (\phi_2(p, m)\ \wedge\ (t1(p, n) < t_2(p, m)\ \vee\ (t_1(p, n) = t_2(p, m) \wedge\ \prec_1 < \prec_2)))$$

L'opération $\oplus$ sur les bornes est utilisée pour l'intersection entre une matrice représentant un ensemble de valuations pour les compteurs et une matrice représentant une garde ou pour la canonisation d'une matrice contrainte. Dans ces deux cas, les variables d'itération apparaissant dans les bornes mises en jeu par l'opération $\oplus$ sont les mêmes. Nous redéfinissons donc l'opération $\oplus$ entre 2 bornes paramétrées ouvertes $\tilde{b}_1 = ((\prec_1, t_1), \phi_1)$ et $\tilde{b}_2 = ((\prec_2, t_2), \phi_2)$ en redéfinissant les formules Φ_i de la manière suivante :

$$\begin{array}{lcl}
\Phi_1 & \equiv & \exists p \in \mathcal{P}\ \exists n \in \mathcal{N}.\ \phi_1(p, n)\ \wedge\ \phi_2(p, n)\ \wedge\ t_1(p, n) < t_2(p, n) \\
\Phi_2 & \equiv & \exists p \in \mathcal{P}\ \exists n \in \mathcal{N}.\ \phi_1(p, n)\ \wedge\ \phi_2(p, n)\ \wedge\ t_1(p, n) = t_2(p, n) \\
\Phi_3 & \equiv & \exists p \in \mathcal{P}\ \exists n \in \mathcal{N}.\ \phi_1(p, n)\ \wedge\ \phi_2(p, n)\ \wedge\ t_1(p, n) > t_2(p, n)
\end{array}$$

L'opération $\otimes$ sur les bornes ouvertes est définie de la même manière que pour les bornes fermées.

Une *matrice de bornes paramétrées ouverte* est, de façon similaire aux bornes, une matrice telle que définie précédemment mais où ses éléments sont des bornes paramétrées ouvertes. Par extension une *matrice de bornes paramétrées contrainte ouverte* est l'association d'une matrice de bornes paramétrées ouverte et d'une formule $\phi \in F(\mathcal{P} \cup \mathcal{N})$ représentant des contraintes sur les paramètres et les variables d'itération.

Les définitions des opérations $\oplus$, $\otimes$, de l'intersection et de la canonisation ne sont pas modifiées par la présence des variables d'itération. Par contre, nous allons devoir redéfinir l'inclusion entre deux matrices ouvertes.

Commençons par étudier un petit exemple. Soient $\{x, y\}$ l'ensemble des compteurs que nous considérons, soient $\{n, m\}$ l'ensemble des variables d'itération. Cet exemple ne contient pas de paramètres. Soient $\psi_1 = x = 4n \ \wedge \ y = 12n$ et $\psi_2 = x = 2m \ \wedge \ y = 3m$ deux formules appartenant à $FEL(\mathcal{N})$. Les matrices ouvertes associées à chacune de ces formules (respectivement $\tilde{M}_1 = (M_1, \phi_1)$ et $\tilde{M}_2 = (M_2, \phi_2)$) sont définies de la manière suivante

$$M_1 = \begin{array}{c} \\ c_0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} c_0 & x & y \end{array} \\ \begin{pmatrix} (\leq, 0) & (\leq, -4n) & (\leq, -12n) \\ (\leq, 4n) & (\leq, 0) & (\leq, -8n) \\ (\leq, 12n) & (\leq, 8n) & (\leq, 0) \end{pmatrix} \end{array} \qquad M_2 = \begin{array}{c} \\ c_0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} c_0 & x & y \end{array} \\ \begin{pmatrix} (\leq, 0) & (\leq, -2m) & (\leq, -3m) \\ (\leq, 2m) & (\leq, 0) & (\leq, -m) \\ (\leq, 3m) & (\leq, m) & (\leq, 0) \end{pmatrix} \end{array}$$

$$\phi_1 = (n \geq 0) \qquad \phi_2 = (m \geq 0)$$

Nous cherchons à savoir si $\tilde{M}_1 \sqsubseteq \tilde{M}_2$. En prenant la définition de l'inclusion donnée dans le paragraphe 4.1.3 (page 58), nous devons décider de la satisfaisabilité des formules suivantes (cases à cases) :

$$\begin{array}{rcl} \forall n \ (n \geq 0) & \Rightarrow & \exists m. \ (m \geq 0 \ \wedge \ -4n \leq -2m) \\ \forall n \ (n \geq 0) & \Rightarrow & \exists m. \ (m \geq 0 \ \wedge \ -12n \leq -3m) \\ \forall n \ (n \geq 0) & \Rightarrow & \exists m. \ (m \geq 0 \ \wedge \ 4n \leq 2m) \\ \forall n \ (n \geq 0) & \Rightarrow & \exists m. \ (m \geq 0 \ \wedge \ -8n \leq -m) \\ \forall n \ (n \geq 0) & \Rightarrow & \exists m. \ (m \geq 0 \ \wedge \ 12n \leq 3m) \\ \forall n \ (n \geq 0) & \Rightarrow & \exists m. \ (m \geq 0 \ \wedge \ 8n \leq m) \end{array}$$

Ces formules prises séparément sont satisfaisables une à une. Pourtant $\tilde{M}_1 \not\sqsubseteq \tilde{M}_2$, en effet prenons la valuation $\overrightarrow{\mathrm{d}} = (x = 4, y = 12)$, nous avons $\overrightarrow{\mathrm{d}} \in [\![\tilde{M}_1]\!]$ mais $\overrightarrow{\mathrm{d}} \notin [\![\tilde{M}_2]\!]$ ce qui démontre que ces ensembles ne sont pas inclus. En fait, dans le cadre des matrices ouvertes, il faut tenir compte du lien qui se crée entre les cases de la matrice. Lorsqu'une valeur pour m est choisie dans la première formule, ce doit être la même valeur qui doit être utilisée dans les formules suivantes (la variable d'itération est affectée à la même valeur pour toute la matrice).

Dans cet exemple, nous voudrions donc, en fait, satisfaire la formule suivante pour prouver l'inclusion :

$$\forall n \ (n \geq 0) \Rightarrow \exists m. \ (m \geq 0 \ \wedge \ 4n = 2m \ \wedge \ 12n = 3m)$$

Cette formule n'est évidemment pas satisfaisable, $\tilde{M_1} \not\sqsubseteq \tilde{M_2}$.

Soient M_1 et M_2 des matrices de bornes paramétrées ouvertes, notons par $M_1(p,n) \leq M_2(p,m)$ la formule

$$\bigwedge_{i=0}^{n_c} \bigwedge_{j=0}^{n_c} (t_{1ij}(p,n) < t_{2ij}(p,m) \ \vee \ (t_{1ij}(p,n) = t_{2ij}(p,m) \wedge \ \prec_{1ij} \leq \prec_{2ij}$$

Soient $\tilde{M_1} = (M_1, \phi_1)$ et $\tilde{M_2} = (M_2, \phi_2)$ deux matrices de bornes paramétrées contraintes ouvertes, $\tilde{M_1} \sqsubseteq \tilde{M_2}$ si et seulement si la formule ϕ_{inc} est satisfaisable.

$$\phi_{inc} \ \equiv \ \forall p \in \mathcal{P} \ \forall n \in \mathcal{N}.\phi_1(p,n) \Rightarrow \exists m \in \mathcal{N}. \ (\phi_2(p,m) \ \wedge \ M_1(p,n) \leq M_2(p,m))$$

Le calcul de l'application d'une transition $post_t$ reste inchangé.

4.3.2 Principe d'accélération

Regardons maintenant comment calculer effectivement en un pas l'ensemble des configurations atteignables par un circuit. Le principe de notre algorithme se décompose en deux parties. Tout d'abord, après avoir détecté automatiquement une boucle, nous *devinons* l'effet de l'itération un nombre arbitraire de fois de cette boucle, puis nous *vérifions* si l'ensemble "deviné" est exact ou non.

Prenons par exemple le système à compteurs composé d'un état de contrôle q_0, d'un compteur x et d'une transition représentée par la figure 4.3(a).

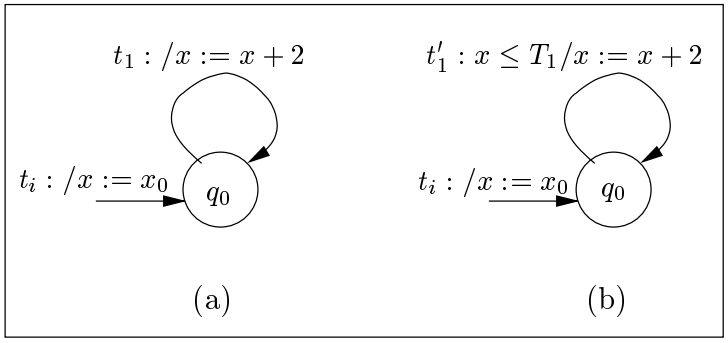


FIG. 4.3 – Un petit exemple

L'idée de l'accélération est de regarder la *distance* parcourue par les variables de notre système en un tour du circuit, puis si la distance est constante nous supposons qu'après un certain nombre de tour n la valeur de la variable sera égale à la valeur initiale + n fois la distance.

Dans notre exemple, supposons que le compteur x ait pour valeur initiale $x = x_0$, après une application de la transition $x = x_0 + 2$, la distance pour le compteur x est alors intuitivement de 2. En réappliquant la transition, le compteur x atteint alors la valeur $x = x_0 + 4$, la deuxième distance est aussi égale à 2. Nous supposons donc que l'ensemble

de toute les valuations possibles pour ce compteur sera $x = 2 * n + x_0$ où n est le nombre d'application du circuit ($n \geq 0$). Bien sûr, si nous rajoutons à la transition une garde par exemple $x \leq T1$ (cf. figure 4.3(b)) il nous faut pouvoir trouver les bonnes conditions sur n car dans ce cas il n'est pas possible d'itérer le circuit un nombre infini de fois. Pour obtenir ces conditions nous verrons qu'il suffit d'appliquer le circuit à partir de la configuration symbolique $(x = 2*n+x_0, n \geq 0)$ ce qui donne l'ensemble $x = 2*n+x_0+2,\ x_0+2*n \leq T_1$.

Plus formellement, voyons tout d'abord comment nous définissons la notion de distance sur les matrices de bornes paramétrées représentant les ensembles de configurations.

La distance entre deux matrices de bornes paramétrées contraintes est représentée par l'opération "$-.$" de la façon suivante : Soient $\tilde{M}_1 = (M_1, \phi_1)$ et $\tilde{M}_2 = (M_2, \phi_2)$ deux matrices contraintes, la distance entre ces deux matrices $\Delta = \tilde{M}_1 -. \tilde{M}_2$ est une matrice carrée de dimension $(n_x + 1) \times (n_x + 1)$ dont les éléments sont des termes arithmétiques sur les paramètres et les variables d'itération ou $\{-\infty, \infty\}$. La distance Δ est telle que :

$$\Delta_{ij} = t_{1ij} - t_{2ij}$$

où $M_{1ij} = (\prec_{1ij}, t_{1ij})$ et $M_{2ij} = (\prec_{2ij}, t_{2ij})$

Nous étendons l'opération $+$ définie précédemment entre une matrice contrainte et une matrice appartenant à Mat^B, pour l'utiliser avec une matrice obtenue par l'opération "$-.$".

Soit Δ une matrice contrainte représentant une distance, nous notons $\overbrace{\Delta + \Delta + \cdots + \Delta}^{\text{n fois}}$ par $n * \Delta$ tel que $n * \Delta[ij] = n * t_{ij}$, n doit être une nouvelle variable d'itération (une variable qui n'a pas encore été utilisée).

Le principe d'accélération est alors le suivant. Si la boucle peut s'exécuter deux fois en obtenant la même distance, l'effet de cette boucle est alors l'ensemble de la configuration initiale plus n fois la distance. Il faut enfin vérifier que la configuration obtenue est bien une configuration atteignable par l'application de la boucle.

Soit $\tilde{M} = (M, \phi)$ une matrice de bornes contrainte, nous notons par $\tilde{M}^+ = (M, \phi \ \wedge \ n \geq 0)$ la matrice contrainte dans laquelle nous avons ajouté une nouvelle variable d'itération n. Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op_g, Op_v, T)$ un automate à compteurs, soit $\theta = t_1 t_2 \cdots t_n$ une boucle de ce système telle que $t_i = (q_i, l_i, o_{gi}, o_{vi}, q_{i+1})$ pour tout $i \in [1, n-1]$ et $t_n = (q_n, l_n, o_{gn}, o_{vn}, q_1)$. Soit $\gamma = (q_1, \tilde{M})$ une configuration atteignable pour l'état de contrôle q_1. Supposons que nous ayons obtenu $(q_1, \tilde{M}_1) \in post_\theta((q_1, \tilde{M}))$ et $(q_1, \tilde{M}_2) \in post_\theta((q_1, \tilde{M}_1))$ deux configurations atteignables à partir de γ en appliquant respectivement une fois et deux fois la boucle. Soit $\Delta = \tilde{M}_1 -. \tilde{M}$ et $\Delta' = \tilde{M}_2 -. \tilde{M}_1$ les deux distances ainsi obtenues. Notre principe d'extrapolation consiste donc en la vérification des deux conditions suivantes :

$$C1 : \ \forall p \in \mathcal{P}\ \forall n \in \mathcal{N}.\ \phi_2(p, n) \Rightarrow \Delta = \Delta'$$
$$C2 : \ \forall n \geq 0.\ post^2_\theta((q, \tilde{M}^+ + n * \Delta)) = post_\theta((q, \tilde{M}^+ + (n+1) * \Delta))$$

La condition $C1$ exprime que l'effet de θ après deux itérations est l'ajout de la distance Δ à chaque pas. Nous remarquons que l'égalité des deux matrices $\Delta = \Delta'$ se décide sous la contrainte ϕ_2 (contrainte associée à la configuration symbolique $(q_1, \tilde{M}_2)$. En effet, cette contrainte est plus forte que la contrainte ϕ_1 (associée à $(q_1, \tilde{M}_1)$), qui est elle-même plus forte que ϕ (associée à $(q_1, \tilde{M})$) car chaque application des transitions de la boucle θ peut éventuellement introduire de nouvelles contraintes sur les paramètres et sur les variables d'itération mais ne peut pas en enlever.

La condition $C2$ quant à elle permet de vérifier que chaque application de θ a pour effet d'ajouter Δ, impliquant que les gardes contenues dans θ sont satisfaites.

Pour tenir compte de ces gardes, nous calculons l'effet après $n+1$ itérations de θ comme la $post_\theta$-image de la configuration $(q_1, \tilde{M}^+ + n\Delta)$. Le lemme suivant permet de dire que si la condition $C2$ est vérifiée nous calculons bien l'ensemble des configurations atteignables du circuit.

Lemme 5

Soit θ un circuit, soit $(q_1, \tilde{M})$ une configuration symbolique et Δ la distance calculée pour ce circuit. Alors les deux formules suivantes sont équivalentes.

1. $\forall n \geq 0.\ post^2_\theta((q, \tilde{M}^+ + n * \Delta)) = post_\theta((q, \tilde{M}^+ + (n+1) * \Delta))$
2. $\forall n \geq 0.\ post^{n+1}_\theta((q_1, \tilde{M})) = post_\theta((q_1, \tilde{M}^+ + n * \Delta))$

Preuve. La preuve se fait par induction sur n.

- $1 \Rightarrow 2$:
  - $(n = 0)$ $post_\theta((q_1, \tilde{M})) = post_\theta((q_1, \tilde{M}^+))$.
  - $(n > 0)$ Supposons que

$$post^{n+1}_\theta((q_1, \tilde{M})) = post_\theta((q_1, \tilde{M}^+ + n * \Delta))$$

et montrons que

$$post^{n+2}_\theta((q_1, \tilde{M})) = post_\theta((q_1, \tilde{M}^+ + (n+1) * \Delta))$$

$$\begin{aligned} post^{n+2}_\theta((q_1, \tilde{M})) &= post_\theta(post^{n+1}_\theta((q_1, \tilde{M})) \\ &= post_\theta(post_\theta((q_1, \tilde{M}^+ + n * \Delta))) \quad (\text{ par Hyp. Ind. }) \\ &= post_\theta((q_1, \tilde{M}^+ + (n+1) * \Delta)) \quad (\text{ par 1. }) \end{aligned}$$

- $2 \Rightarrow 1$:
  - $(n = 0)$ $post^2_\theta((q_1, \tilde{M}^+)) = post_\theta((q_1, \tilde{M}^+ + \Delta))$ par 2..
  - $(n > 0)$ Supposons que

$$post^2_\theta((q_1, \tilde{M}^+ + n * \Delta)) = post_\theta((q_1, \tilde{M}^+ + (n+1) * \Delta))$$

et montrons que

$$post^2_\theta((q_1, \tilde{M}^+ + (n+1) * \Delta)) = post_\theta((q_1, \tilde{M}^+ + (n+2) * \Delta))$$

$post^2_\theta((q_1, \tilde{M}^+ + (n+1) * \Delta))$

$$
\begin{aligned}
&= post_\theta(post_\theta((q_1, \tilde{M}^+ + (n+1) * \Delta))) \\
&= post_\theta(post_\theta^2((q_1, \tilde{M}^+ + n * \Delta))) && (\text{ par Hyp. Ind. }) \\
&= post_\theta(post_\theta(post_\theta((q_1, \tilde{M}^+ + n * \Delta)))) \\
&= post_\theta(post_\theta(post_\theta^{n+1}((q_1, \tilde{M})))) && (\text{ par 2. }) \\
&= post_\theta^{n+3}((q_1, \tilde{M})) \\
&= post_\theta((q_1, \tilde{M}^+ + (n+2) * \Delta)) && (\text{ par 2. })
\end{aligned}
$$

■

Nous venons de calculer l'ensemble des configurations atteignables par itération d'un circuit, nous avons alors :

$$post^*((q_1, \tilde{M})) = post_\theta(\tilde{M}^+ + n * \Delta)$$

Ainsi, si les conditions d'extrapolation définies ci-dessus sont appliquables, l'ensemble des configurations symboliques $post_\theta((q_1, \tilde{M}^+ + n * \Delta))$ est ajouté à l'ensemble des configurations atteignables.

Regardons maintenant les effets de l'accélération sur le graphe symbolique. La figure 4.4 représente l'état du graphe avant l'accélération (a) et après l'accélération (b). Avant l'accélération, l'algorithme de calcul des configurations atteignables en profondeur trouve une boucle θ entre $(q_1, \tilde{M})$ et $(q_1, \tilde{M}_1)$. Si le principe d'extrapolation s'applique sur cette boucle, nous pouvons alors calculer $post_\theta((q_1, \tilde{M}^+ + n * \Delta))$ qui sont les configurations atteignables par le circuit. Si nous remarquons que la configuration $(q_1, \tilde{M}_1)$ est contenue dans $post_\theta((q_1, \tilde{M}^+ + n * \Delta))$, nous modifions alors le graphe symbolique de la manière suivante :

- nous échangeons la configuration $(q_1, \tilde{M}_1)$ par $post_\theta((q_1, \tilde{M}^+ + n * \Delta))$.
- nous appliquons le circuit à partir de $post_\theta((q_1, \tilde{M}^+ + n * \Delta))$, nous obtenons ainsi une boucle dans le graphe symbolique sur $post_\theta((q_1, \tilde{M}^+ + n * \Delta))$.

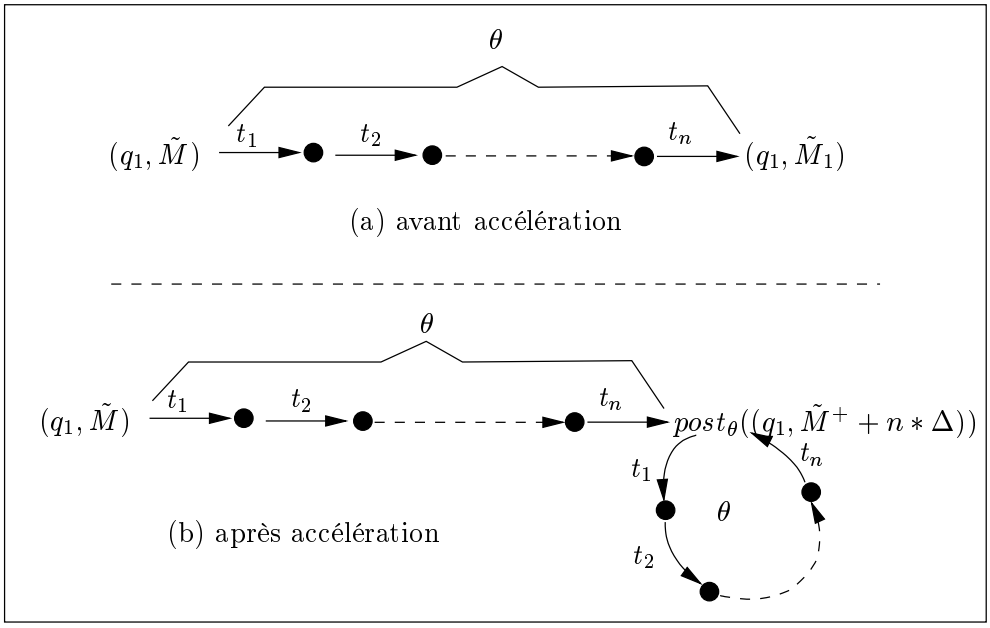


FIG. 4.4 – Représentation du graphe symbolique

4.4 Problèmes de décision

Nous avons introduit les notions d'arithmétique dans la section 2.1.2 (page 8). Les formules dont nous devons décider de la satisfaisabilité sont de trois sortes :

$$\begin{array}{ll} (1) & \forall p \in \mathcal{P}.\forall n \exists m \in \mathcal{N}.\phi \\ (2) & \forall p \in \mathcal{P}.\forall n \in \mathcal{N}.\phi \\ (3) & \exists p \in \mathcal{P}.\exists n \in \mathcal{N}.\phi \end{array}$$

Nous distinguons alors plusieurs cas :

- *Les paramètres sont tous des entiers.*

  Dans ce cas, si ϕ est linéaire, les trois formules sont des formules de l'arithmétique de Presburger et sont alors décidables.

  Par contre, s'il existe au moins un terme non linéaire dans ϕ alors nous ne pouvons pas conclure sur la satisfaisabilité des formules. Dans ce dernier cas, nous avons un test sauf pour la formule (2). En effet, nous pouvons décider de la satisfaisabilité de cette formule en considérant les variables comme réelles. Si (2) est insatisfaisable dans $FR(\mathcal{P} \cup \mathcal{X})$, alors elle est aussi insatisfaisable pour les entiers. Aussi, nous sommes sûrs que si la formule (2) n'est pas satisfaisable dans les réels elle ne l'est pas non plus pour les entiers. D'un autre côté si la formule est satisfaisable dans les réels cela ne prouve pas qu'elle le soit pour les entiers.
- *Les paramètres sont tous des réels :*

  Deux cas se présentent :
  - *La formule ne contient pas de variables d'itération.*

    Dans ce cas, les trois formules sont décidables (même dans le cas non linéaire).
  - *La formule contient des variables d'itération.*

    Dans ce cas, nous sommes dans ce que nous avons appelé *arithmétique demi-linéaire*. Pour analyser ces formules nous appliquons le principe suivant : nous éliminons les paramètres réels de la formule. Ce qui nous donne une formule ne contenant plus que des variables entières. Si la formule ainsi obtenue est linéaire sur les entiers nous pouvons alors décider. Si ce n'est pas le cas, nous ne pourrons rien dire.
- *Les paramètres sont des réels et des entiers :*

  Les formules appartiennent alors à l'arithmétique demi-linéaire. La méthode décrite ci-dessus s'applique à ce cas.

Remarque : Dans le cadre de l'arithmétique demi-linéaire il nous faut faire attention à l'ordre dans lequel nous éliminons les variables réelles. En effet, suivant l'ordre choisi la formule résultante peut être linéaire ou non linéaire. Nous discutons plus en détail de ce point dans la section 7.2.2.1 (page 122).

4.5 Exemples

4.5.1 Systèmes à compteurs paramétrés : Algorithme du "Bakery"

Ainsi, nous venons de définir une représentation pour les valuations des compteurs et des paramètres, nous avons défini les opérations *post* et $post^*$ sur cette représentation. Nous pouvons les utiliser et appliquer l'algorithme de calcul en profondeur des configurations atteignables 3.4 pour l'exemple du "bakery". Nous avons introduit cet exemple à la page 14.

Dans cet exemple, la configuration initiale est $\gamma_0 = ((repos_1, repos_2), c_1 = 0 \wedge c_2 = 0)$.

L'ensemble des configurations symboliques atteignables est représenté dans le tableau 4.5.

Le graphe symbolique (construit par l'analyse) est représenté par la figure 4.6.

état	a^1	a^2	contrainte
$repos_1\ repos_2$	0	0	
$repos_1\ attente_2$	0	1	
	0	2	
	0	$4+2*n_0$	$n_0 \geq 0$
	0	$3+2*n_1$	$n_1 \geq 0$
$repos_1\ sectionc_2$	0	1	
	0	2	
	0	$4+2*n_0$	$n_0 \geq 0$
	0	$3+2*n_1$	$n_1 \geq 0$
$attente_1\ repos_2$	1	0	
	2	0	
	$3+2*n_0$	0	$n_0 \geq 0$
	$4+2*n_1$	0	$n_1 \geq 0$
$attente_1\ attente_2$	1	2	
	2	1	
	2	3	
	3	2	
	$3+2*n_0$	$4+2*n_0$	$n_0 \geq 0$
	$4+2*n_1$	$3+2*n_1$	$n_1 \geq 0$
	$5+2*n_0$	$4+2*n_0$	$n_0 \geq 0$
	$4+2*n_1$	$5+2*n_1$	$n_1 \geq 0$
$attente_1\ sectionc_2$	2	1	
	3	2	
	$5+2*n_0$	$4+2*n_0$	$n_0 \geq 0$
	$4+2*n_1$	$3+2*n_1$	$n_1 \geq 0$
$sectionc_1\ repos_2$	1	0	
	2	0	
	$3+2*n_0$	0	$n_0 \geq 0$
	$4+2*n_1$	0	$n_1 \geq 0$
$sectionc_1\ attente_2$	1	2	
	2	3	
	$3+2*n_0$	$4+2*n_0$	$n_0 \geq 0$
	$4+2*n_1$	$5+2*n_1$	$n_1 \geq 0$

FIG. 4.5 – Ensemble des configurations atteignables du "Bakery"

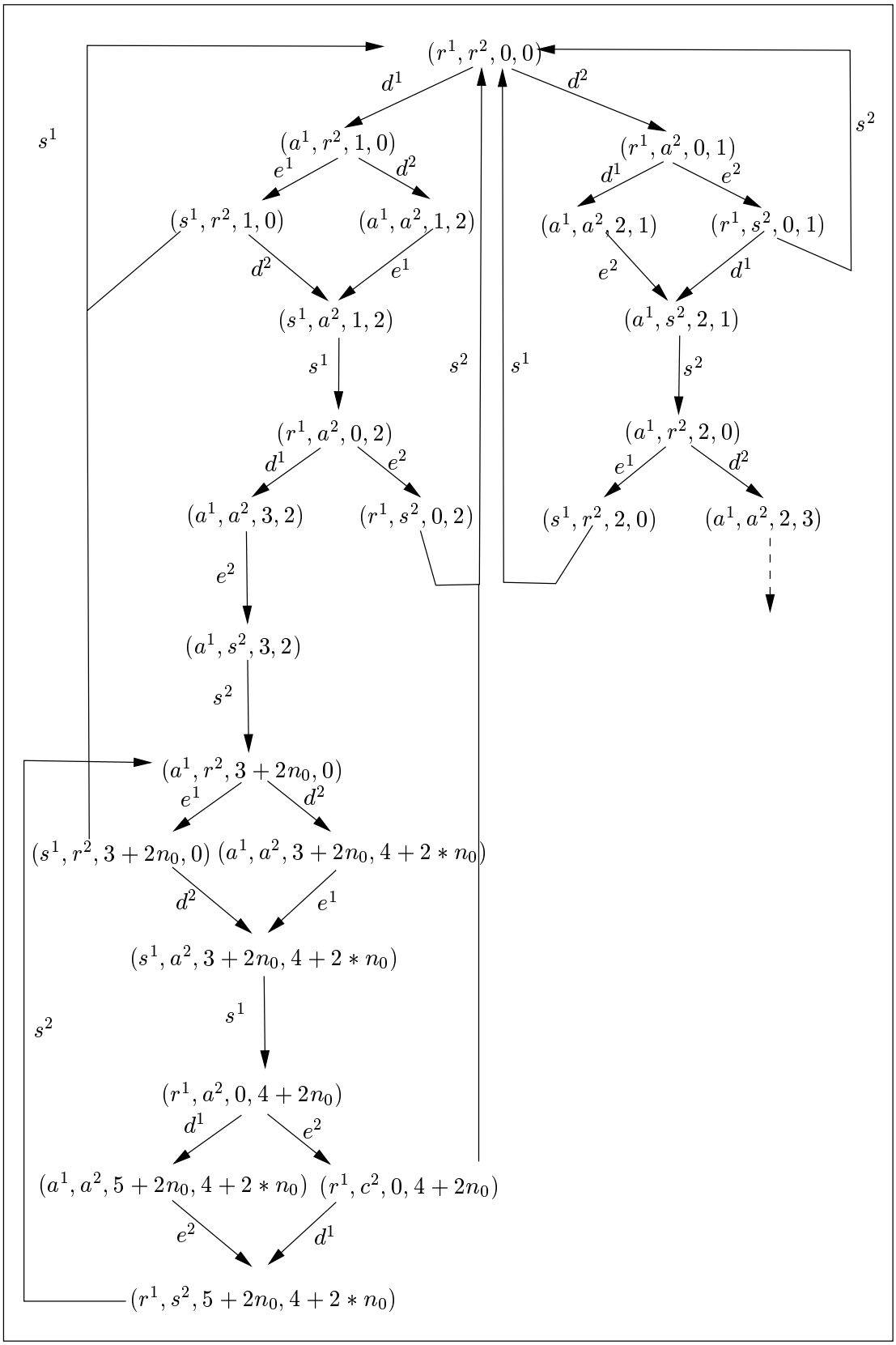


FIG. 4.6 – Graphe symbolique du “Bakery”

4.5.2 Systèmes à horloges paramétrées : Algorithme du "Fisher"

Appliquons l'algorithme de calcul des configurations atteignables sur l'exemple du "Fisher". (voir section 2.4.2.2, page 18 pour la description de cet exemple).

La configuration initiale est $\gamma_0 = ((repos^1, repos^2),\ 0 \leq h_1 \ \wedge\ 0 \leq h_2 \ \wedge\ h_1 = h_2,\ T > 0 \ \wedge\ D > 0 \ \wedge\ T \geq D$.

L'ensemble des configurations symboliques atteignables est présenté dans le tableau 4.7 où :

$$\phi_1 = T > 0 \ \wedge\ D \leq T \ \wedge\ D > 0$$

$$\phi_2 = T = D \ \wedge\ D > 0$$

$$\phi_3 = D < T \ \wedge\ D > 0$$

Les cases du tableau contenant plusieurs lignes doivent être lues comme une disjonction sur les valeurs des horloges et des paramètres.

Le graphe symbolique est représenté par la figure 4.8 où les états de contrôle reprennent les numéros des configurations apparaissant dans le tableau ci-dessus.

	état	id	$\tilde{M}$	contrainte
γ_0	$repos^1\ repos^2$	0	$h_1 > T,\ h_2 \geq 0,\ h_2 - h_1 \leq 0$	ϕ_2
			$h_1 > T,\ h_2 \geq 0,\ h_2 - h_1 < D - T$	ϕ_3
			$h_1 > T,\ h_2 \geq 0,\ h_1 - h_2 < T$	ϕ_1
			$h_1 \geq 0,\ h_2 > T,\ h_2 - h_1 < T$	ϕ_1
			$h_1 > T,\ h_2 > T,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
			$h_1 \geq 0,\ h_2 > T,\ h_1 - h_2 \leq 0$	ϕ_2
			$h_1 \geq 0,\ h_2 > T,\ h_1 - h_2 < D - T$	ϕ_3
			$h_1 > T,\ h_2 > T,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
			$h_1 \geq 0,\ h_2 \geq 0,\ h_1 = h_2$	ϕ_1
γ_1	$sessiond^1\ repos^2$	0	$0 \leq h_1 \leq D,\ 0 \leq h_2 \leq D,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
			$0 \leq h_1 \leq D,\ 0 \leq h_2,\ 0 \leq h_2 - h_1$	ϕ_1
γ_2	$attente^1\ repos^2$	0	$h_1 > T,\ h_2 > T,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
γ_3	$attente^1\ repos^2$	1	$h_1 \geq 0,\ h_2 \geq 0,\ h_2 - h_1 \leq 0$	ϕ_2
			$0 \leq h_1,\ 0 \leq h_2,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
			$h_1 > T - D,\ 0 \leq h_2,\ h_2 - h_1 < D - T$	ϕ_3
γ_4	$sectionc^1\ repos^2$	1	$h_1 > T,\ h_2 \geq 0,\ h_2 - h_1 \leq 0$	ϕ_2
			$h_1 > T,\ 0 \leq h_2,\ h_2 - h_1 \leq D - T$	ϕ_3
			$h_1 > T,\ h_2 \geq 0,\ 0 \leq h_2 - h_1 < -T$	ϕ_1
			$h_1 > T,\ h_2 > T,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
γ_5	$repos^1\ sessiond^2$	0	$0 \leq h_1 \leq D,\ 0 \leq h_2 \leq D,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
			$0 \leq h_1,\ 0 \leq h_2 \leq D,\ h_2 - h_1 \leq 0$	ϕ_1
γ_6	$sessiond^1\ sessiond^2$	0	$0 \leq h_1 \leq D,\ 0 \leq h_2 \leq D,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
			$0 \leq h_1 \leq D,\ 0 \leq h_2 \leq D,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
γ_7	$attente^1\ sessiond^2$	0	$T < h_1, 0 \leq h_2 \leq D,\ h_2 - h_1 < -T$	ϕ_1
γ_8	$attente^1\ sessiond^2$	1	$0 \leq h_1,\ 0 \leq h_2 \leq D,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
γ_9	$repos^1\ attente^2$	0	$h_1 > T,\ h_2 > T,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
γ_{10}	$repos^1\ attente^2$	2	$0 \leq h_1,\ 0 \leq h_2 \leq D,\ h_1 - h_2 \leq 0$	ϕ_2
			$0 \leq h_1,\ 0 \leq h_2,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
			$0 \leq h_1,\ T - D < h_2,\ h_1 - h_2 < D - T$	ϕ_3
γ_{11}	$sessiond^1\ attente^2$	0	$T < h_2,\ 0 \leq h_1 \leq D,\ h_1 - h_2 < -T$	ϕ_1
γ_{12}	$sessiond^1\ attente^2$	2	$0 \leq h_1 \leq D,\ 0 \leq h_2 \leq D,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
γ_{13}	$attente^1\ attente^2$	2	$0 \leq h_1,\ 0 \leq h_2,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
γ_{14}	$attente^1\ attente^2$	1	$0 \leq h_1,\ 0 \leq h_2,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
γ_{15}	$sectionc^1\ attente^2$	1	$h_1 > T,\ h_2 > T,\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
γ_{16}	$repos^1\ sectionc^2$	2	$0 \leq h_1,\ T < h_2,\ h_1 - h_2 < -T$	ϕ_1
			$h_1 > T,\ h_2 > T,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
			$0 \leq h_1,\ T < h_2,\ h_1 - h_2 < D - T$	ϕ_3
			$0 \leq h_1,\ T < h_2,\ h_1 - h_2 \leq 0$	ϕ_2
γ_{17}	$attente^1\ sectionc^2$	2	$h_1 > T,\ h_2 > T,\ 0 \leq h_1 - h_2 \leq D$	ϕ_1

FIG. 4.7 – Ensemble des configurations atteignables du “Fisher”

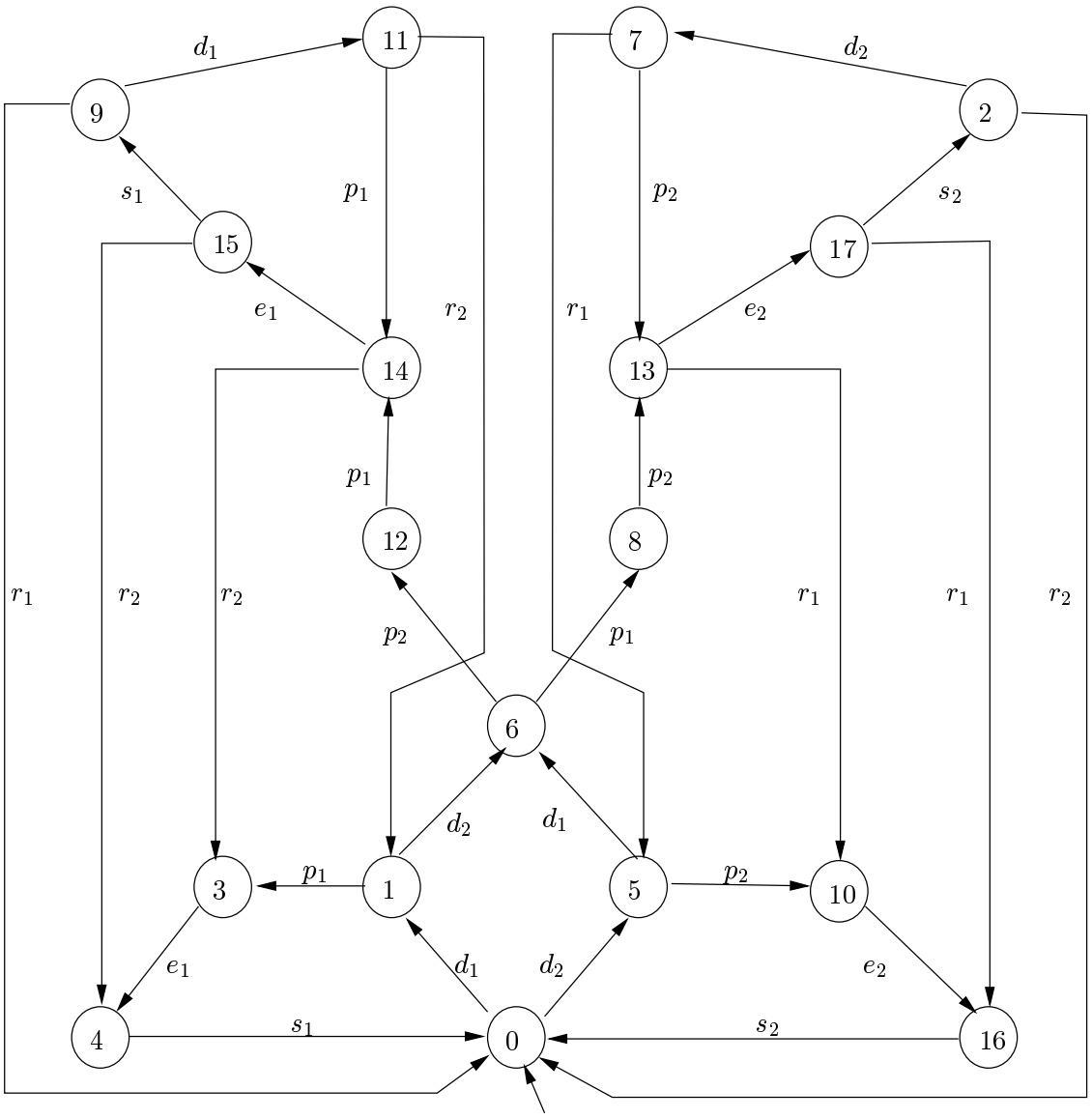


FIG. 4.8 – Graphe symbolique du "Fisher"

4.5.3 Systèmes hétérogènes

Dans le chapitre 3, nous avons discuté de la composition de système. Nous avons vu qu'il nous fallait soit trouver une représentation générale pour les différents domaines, soit deux représentations avec la possibilité de garder le lien des itérations de circuits.

Voici deux exemples illustrant les deux techniques proposées.

Nous reprenons tout d'abord l'exemple donné dans l'introduction de ce chapitre en supposant que le reste du système n'affecte pas l'horloge h ni le compteur c.

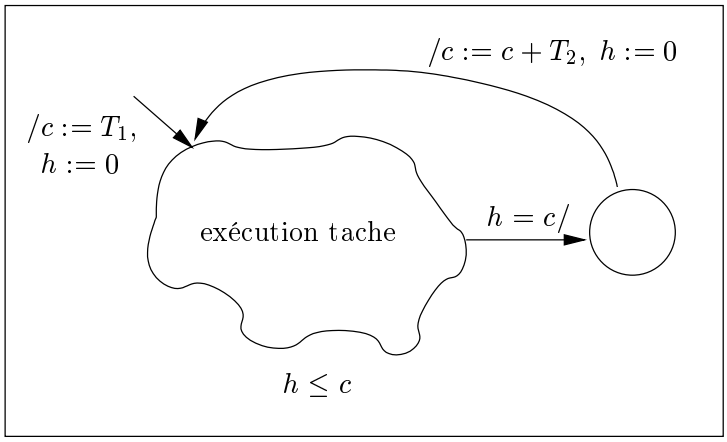


FIG. 4.9 – Un exemple de composition

La configuration initiale est la suivante :

$$\begin{pmatrix} & x_0 & h & c \\ x_0 & (\leq, 0) & (\leq, 0) & (\leq, -T_1) \\ h & (\leq, T_1) & (\leq, 0) & (\leq, 0) \\ c & (\leq, T_1) & (\leq, T_1) & (\leq, 0) \end{pmatrix}$$

Il est possible de tester la garde $h = c$, d'ajouter T_2 au compteur et de remettre à zéro l'horloge.

Si nous calculons les configurations obtenues après une application et deux applications du circuit, nous obtenons la distance suivante :

$$\begin{pmatrix} 0 & 0 & -T_2 \\ T_2 & 0 & 0 \\ T_2 & T_2 & 0 \end{pmatrix}$$

Nous appliquons alors notre principe d'accélération, ce qui donne l'ensemble final des configurations suivant :

$$\begin{pmatrix} & x_0 & h & c \\ x_0 & (\leq, 0) & (\leq, 0) & (\leq, -T_1 - (n+1)T_2) \\ h & (\leq, T_1 + (n+1)T_2) & (\leq, 0) & (\leq, 0) \\ c & (\leq, T_1 + (n+1)T_2) & (\leq, T_1 + (n+1)T_2) & (\leq, 0) \end{pmatrix}$$

Ainsi, d'avoir gardé le lien entre le compteur et l'horloge nous a permis de pouvoir analyser ce système. De la même manière, nous sommes capables d'analyser tout système dont les transitions sont de la forme $x := y + t$ où x et y sont respectivement soit une horloge, soit un compteur.

Observons maintenant un détail de l'exemple du protocole de retransmission bornée. Cet exemple fait intervenir deux horloges x et y et un compteur c. La figure 4.10 représente le moment où l'émetteur réemet le message qu'il est en train d'envoyer tant qu'il n'a pas reçu d'acquittement, dans une limite de max fois.

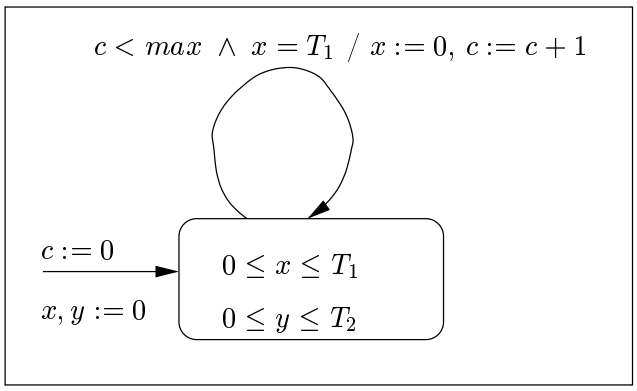


FIG. 4.10 – BRP — la réémission

Pour cet exemple, il n'est pas nécessaire de conserver les liens entre les valeurs des compteurs et des horloges. Seul le lien afférant au nombre d'exécution de la boucle est nécessaire. Nous pouvons donc séparer la représentation des compteurs et des horloges. Nous avons choisi de représenter les valuations des horloges par une matrice de bornes paramétrées, les valuations des compteurs par une autre matrice et les liens entre les paramètres sont représentés par une formule arithmétique.

La configuration initiale est :

$$\left(\begin{pmatrix} & x_0 & x & y \\ x_0 & (\leq,0) & (\leq,0) & (\leq,0) \\ x & (\leq,T_1) & (\leq,0) & (\leq,0) \\ y & (\leq,T_1) & (\leq,0) & (\leq,0) \end{pmatrix} \quad \begin{pmatrix} & x_0 & c \\ x_0 & (\leq,0) & (\leq,0) \\ c & (\leq,0) & (\leq,0) \end{pmatrix} \right.$$
$$\left. 2 * max * T_1 \leq T_2 \ \wedge \ T_1 \geq 0 \ \wedge \ T_2 \geq 0 \ \right)$$

La distance pour les horloges est de :

$$\begin{pmatrix} 0 & 0 & -T_1 \\ 0 & 0 & -T_1 \\ T_1 & T_1 & 0 \end{pmatrix}$$

Celle pour les compteurs est :

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

Nous pouvons appliquer notre principe d'accélération et nous obtenons

$$\left(\begin{pmatrix} (\leq,0) & (\leq,0) & (\leq,-(n_0+1)T_1) \\ (\leq,T_1) & (\leq,0) & (\leq,-(n_0+1)T_1) \\ (\leq,(n_0+2)T_1) & (\leq,(n_0+1)T_1) & (\leq,0) \end{pmatrix} \quad \begin{pmatrix} (\leq,0) & (\leq,-(n_1+1)) \\ (\leq,n_1+1) & (\leq,0) \end{pmatrix} \right.$$

$$\left. 2 * max * T_1 \leq T_2 \ \wedge \ T_1 \geq 0 \ \wedge \ T_2 \geq 0 \ \wedge \ n_1 + 1 \leq max \ \wedge \ n_0 = n_1 \ \wedge \ 2T_1 + n_0 * T_1 < T_2 \ \right)$$

4.6 Discussion

A notre connaissance, il n'existe pas de méthode et de représentation d'ensemble de compteurs ou d'horloges permettant de rester exact et de pouvoir manipuler des paramètres ainsi que des formules non linéaires.

Pour les compteurs la plupart des représentations sont basées sur la notion de formules de Presburger. Il existe des travaux qui utilisent directement les formules [Bul98] ou bien un sous-ensemble tel que les formules exprimées par la contrainte logique [DP99]. Dans ces deux travaux, la terminaison de l'algorithme de calcul des états atteignables est aidée par des techniques d'élargissement. Le résultat fourni est alors une sur-approximation de l'ensemble des configurations atteignables.

Nous trouvons aussi des représentations sous forme d'automates : Numerical Decision Diagram [Boi98]. La méthode d'analyse est basée sur la détection de périodicité dans les transitions afin de permettre de donner (si l'ensemble résultat reste linéaire) l'ensemble des configurations atteignables par tout circuit périodique [BW94].

Pour les systèmes temporisés, les méthodes d'analyse n'utilisent pas d'élargissement ou d'accélération car l'ensemble des régions atteignables (une région est un polyhèdre convexe représenté par une matrice à différence de bornes) pour un système sans paramètres est fini.

Si nous nous penchons sur les différentes représentations, nous constatons tout d'abord que les matrices à différence de bornes (sans paramètres) définies par [MB83, Dil89] ont été largement utilisées dans des outils comme Kronos [DOTY96, Yov97] ou Uppaal [LPY97, LP00b, LP00a].

Le problème des matrices est qu'il n'est pas possible de faire l'union sans approximer, aussi deux représentations ont été développées pour pallier à ce problème. Sur le principe des Binary Decision Diagram [Bry92] (représentation compacte pour les formules booléennes), Moller et Lichtenberg ont développé les Difference Decision Diagram [MLAH99, ML99]. Parallèlement, Larsen et Yi ont développé les Clock Difference Diagram [LWYP98, BLP+99] qui sont maintenant utilisées dans Uppaal. Aucune de ces représentations ne permet la manipulation de paramètres.

Parallèlement à notre travail, Vaandrager et al. [HRSV01] ont développé pour Uppaal une extension des matrices de bornes pouvant manipuler des paramètres. Le modèle utilisé ne permet pas l'analyse de modèle contenant des gardes de la forme $x = T$, ni de prendre en compte les effets non linéaires. Nous détaillons ces travaux dans le chapitre sur notre outil (chapitre 7).

L'outil Hytech [HHWT95, HH95] quant à lui, utilise les polyhèdres [Hal93, HRP94] comme représentation des ensembles de valuations sur les horloges et permet l'utilisation de paramètres. La méthode d'analyse utilisée ne contient pas d'accélération, il n'est alors pas possible d'analyser des systèmes avec plusieurs paramètres tels que l'exemple du détail du BRP.

Boigelot, Rassart et Wolper ont développé une représentation des vecteurs de réels par un automate de Büchi non déterministe [BBR97, BRW98]. Cette représentation sous forme d'automates permet de manipuler en même temps des entiers et des réels. [BJW01] ont montré que les formules d'arithmétiques linéaires pour les entiers et les réels sont décidables en représentant les formules par des automates. Ces représentations ne permettent pas de représenter des formules non linéaires. Nous ne pouvons alors pas représenter l'ensemble des configurations atteignables par accélération pour les horloges.

Chapitre 5

Vérification de propriétés de vivacité

L'approche décrite pour les propriétés de sûreté ne fonctionne pas, en général, pour la vérification de propriétés de vivacité. En effet, la plupart du temps, les propriétés de vivacité ne peuvent être vérifiées que sous des conditions d'équité ("fairness" en anglais) précisant que les itérations infinies de certaines transitions ne sont pas possibles.

L'idée est alors d'obtenir une abstraction équitable du modèle initial à partir de laquelle nous pouvons utiliser des méthodes et des outils de vérification de propriétés de vivacité sur des systèmes finis tel que SMV [Mac93].

La vérification de propriétés de vivacité est donc effectuée sur des systèmes de transitions étiquetés contenant des conditions d'équité sur les transitions. La question qui se pose alors est de savoir comment introduire automatiquement ces conditions sur le modèle fini abstrait obtenu par l'analyse pour être capable de vérifier les propriétés de vivacité. Ces conditions sont de deux sortes :

- Soit, nous connaissons à priori les conditions d'équité sur le système initial, nous devons alors étendre le modèle des automates étendus pour tenir compte de ces conditions. Puis nous devons transférer ces conditions du modèle initial au graphe symbolique.
- Soit, nous allons ajouter des conditions d'équité (les calculer automatiquement) directement sur le graphe symbolique. En effet, en général, les boucles qui se trouvent dans le graphe d'espace des états ne correspondent pas à une exécution infinie dans le modèle initial.

Nous commençons dans ce chapitre par étendre le modèle des automates étendus avec des notions de contraintes d'équité. Nous expliquons ensuite comment passer ces contraintes de l'automate à sa sémantique (nous redéfinissons pour cela la notion de $post_t$) puis nous montrons la méthode générale pour obtenir des contraintes supplémentaires pour le graphe symbolique.

Nous traitons ensuite du cas particulier des compteurs et des horloges.

Les résultats décrits dans ce chapitre ont été publiés dans l'article [BCALS01].

5.1 Modèle des automates étendus équitables

5.1.1 Syntaxe

Un *automate étendu équitable* $\mathcal{A}_e$ est un automate étendu $\mathcal{A} = (Q,\ \mathcal{X},\ \mathcal{P},\ L,\ Op,\ T)$ sur lequel nous avons ajouté des contraintes d'équité. Nous ne considérons ici que deux sortes de conditions d'équité :

- les conditions d'équité faible : elles expriment le fait que si lors de l'exécution il est toujours possible de prendre une transition (les conditions de la garde sont vérifiées sur toutes les configurations de l'exécution à partir d'un certain point) alors cette transition sera inévitablement prise.
- les conditions d'itérabilité bornée : elles expriment que l'itération d'une boucle dans le graphe de l'espace d'états ne peut pas être prise infiniment souvent.

Avant de définir formellement les automates équitables nous introduisons la notion d'*action contrainte*. Une action contrainte est un couple $(t,\ \phi)$ où t est une transition de l'automate $\mathcal{A}$ et $\phi \in F(\mathcal{X} \cup \mathcal{P})$ est une formule exprimant une contrainte sur les valeurs sources des variables pour la transition t.

Soit $\mathcal{U}$ un ensemble d'actions contraintes, $T(\mathcal{U})$ représente l'ensemble des transitions qui apparaissent dans $\mathcal{U}$.

Définition 29 (Automate étendu équitable)
Soit $\mathcal{A} = (Q,\ \mathcal{X},\ \mathcal{P},\ L,\ Op,\ T)$ *un automate étendu équitable.* $\mathcal{A}_e = (Q,\ \mathcal{X},\ \mathcal{P},\ L,\ Op,\ T,\ \mathcal{F},\ \mathcal{I}_\mathcal{B})$ *est un automate étendu équitable si et seulement si soit* $\mathcal{S}(\mathcal{A})$ *le graphe d'espace des états de* $\mathcal{A}$*, pour toutes séquences d'exécution* σ *de* $\mathcal{S}(\mathcal{A})$ *nous avons :*

- *l'ensemble* $\mathcal{F}$ *est un ensemble d'actions contraintes, nommé* ensemble d'équité. *Il exprime des conditions dites faibles : si une action de cet ensemble est possible en permanence, la transition correspondante doit être exécutée inévitablement dans le futur.*
  Plus formellement, la séquence d'exécution $\sigma = \gamma_0 t_0 \gamma_1 t_1 \cdots$ *est une séquence équitable si et seulement si :*

$$\forall (t, \phi) \in \mathcal{F}.\ \forall i \geq 0,\ si\ \forall j \geq i.\ \gamma_j \models \phi\ alors\ \exists k \geq j.\ t_k = t$$

- *l'ensemble* $\mathcal{I}_\mathcal{B}$ *est un ensemble d'ensembles d'actions contraintes, nommé* ensemble d'Itérabilité Bornée. *Il exprime des conditions d'itérabilité d'un ensemble de transitions : pour chaque ensemble* $\mathcal{U}$ *de* $\mathcal{I}_\mathcal{B}$*, si toutes les actions de* $\mathcal{U}$ *sont infiniment souvent appliquées alors il doit exister infiniment souvent une action n'appartenant pas à* $\mathcal{U}$ *qui est prise.*
  Plus formellement, la séquence d'exécution $\sigma = \gamma_0 t_0 \gamma_1 t_1 \cdots$ *est une séquence équitable si et seulement si :*

$$\forall \mathcal{U} \in \mathcal{I}_\mathcal{B}.\ si\ \forall (t, \phi) \in \mathcal{U}\ \exists^\omega j.\ (t_j = t\ \wedge\ \gamma_j \models \phi)\ alors\ \exists t' \notin T(\mathcal{U})\ \exists^\omega k.\ t' = t_k$$

où $\exists^\omega$ *s'énonce "il existe infiniment souvent".*

5.1.2 Sémantique

Là encore, le but de l'analyse des automates équitables est de construire le graphe de l'espace des états représentant sa sémantique sur lequel il sera possible d'appliquer des méthodes de model-checking pour vérifier la propriété. Dans le cadre où nous nous plaçons, nous devons redéfinir la notion de graphe de l'espace des états pour lui ajouter des conditions d'équité. Nous commençons par définir le parallèle des actions contraintes pour le graphe de l'espace des états $\mathcal{S}(\mathcal{A}) = (C, L, R)$ que nous nommons *arête contrainte*. Une arête contrainte est un couple (r, b) où $r \in R$ est une transition du graphe de l'espace des états et $b \in \mathbb{B}$ est un booléen exprimant une contrainte sur la configuration qui est dans la source de la transition.

Définition 30 (Graphe symbolique équitable)
Soit $\mathcal{A}_e$ un automate étendu équitable. $\mathcal{S}(\mathcal{A}) = (C, L, R, \mathcal{F}_g, \mathcal{I}_{\mathcal{B}g})$ représente la sémantique équitable de $\mathcal{A}$ telle que :
- *(C, L, R) est le graphe symbolique de l'automate sans les contraintes d'équité.*
- *$\mathcal{F}_g$ est un ensemble d'arêtes contraintes, il est appelé ensemble d'équité.*
- *$\mathcal{I}_{\mathcal{B}g}$ est un ensemble d'ensembles d'arêtes contraintes, il est appelé ensemble d'itérabilité bornée.*

Nous voyons dans les paragraphes suivants comment remplir les ensembles d'équité et d'itération bornée pour le graphe de l'espace des états en utilisant les indications données par les ensembles d'équité et d'itération bornée de l'automate $\mathcal{A}$.

5.1.2.1 Transfert de conditions d'équité

Nous voyons dans cette partie comment translater les conditions d'équité exprimées dans l'automate de départ sur le graphe .

Pour commencer, considérons l'ensemble des actions contraintes (t_i, ϕ_i) de l'automate $\mathcal{A}$ où $i \in [1, l]$ tel que l soit le nombre d'actions contraintes contenues dans $\mathcal{A}$. Nous associons à chaque formule ϕ_i un booléen b_i.

Pour pouvoir effectivement tenir compte des contraintes d'équité, il faut savoir si chaque configuration donnée satisfait ou non une ou plusieurs formules contenues dans les actions contraintes. Pour cela, nous étendons la notion de configuration $\gamma = (q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$ en ajoutant une valuation ν pour les booléens b_i ($\nu \in \mathbb{B}^l$). Les configurations que nous considérons maintenant sont donc l'association d'un état de contrôle q, d'une valuation $\overrightarrow{\mathrm{d}}$ sur les variables de l'automate, d'une valuation $\overrightarrow{\mathrm{p}}$ sur les paramètres et d'une valuation ν pour les booléens b_i : $\gamma^f = (q, \overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}, \nu)$. Nous remarquons que si $\nu(b_i)$ est vrai (resp. faux) alors la configuration satisfait ϕ_i (resp. ne satisfait pas ϕ_i).

Nous avons calculé l'ensemble des configurations étendues atteignables par l'application d'une transition, regardons maintenant comment transposer les contraintes d'équité.

Commençons par les contraintes d'équité faible. Supposons que (t, ϕ) soit une action contrainte de l'ensemble $\mathcal{F}$. Si nous voulons vérifier une propriété de vivacité sur le graphe de l'espace des états, nous ne devrons considérer que les chemins qui contiennent infiniment souvent la transition t et si à partir d'un nœud de ce graphe, toutes les configurations étendues qui appartiennent à ce chemin sont telles que le booléen associé à la formule ϕ

soit vrai. En d'autres termes, l'ensemble $\mathcal{F}_g$ devra contenir l'arête contrainte (r_t, b_ϕ) où r_t représente les transitions étiquetées par t et b_ϕ est le booléen associé à la formule ϕ.

Nous pouvons appliquer le même raisonnement aux contraintes d'itérabilité bornée. Soit $\mathcal{U} = \{(t_i, \phi_i) | 1 \leq i \leq n\} \in \mathcal{I}_\mathcal{B}$ un ensemble appartenant à l'ensemble d'itérabilité bornée. Pour transposer ces contraintes dans le graphe de l'espace des états équitable, il faut que l'ensemble $\{(r_{t_i}, \vee_{i=1}^{n} b_i) \mid 1 \leq i \leq n\}$ soit inclus dans l'ensemble $\mathcal{I}_{\mathcal{B}g}$.

5.1.2.2 Générer des contraintes d'équité

Comme nous l'avons expliqué au début de cette partie, transposer les contraintes d'équité contenues dans l'automate de départ n'est pas suffisant à l'analyse de propriété de vivacité. Il se peut, en effet, que dans le graphe de l'espace des états équitables il existe des chemins infinis qui ne correspondent à aucune exécution réelle de l'automate équitable original. Ce sont typiquement des chemins qui correspondent à l'itération infinie de boucle qui ne peuvent être exécutées un nombre infini de fois dans le modèle original sans transgresser des gardes. Pour cette raison, nous voulons non seulement construire le graphe de l'espace des états mais aussi analyser l'itérabilité des boucles que ces dernières soient simples ou complexes (c'est-à-dire analyser toutes les composantes connexes du graphe). L'itérabilité d'une boucle est une fonction qui appliquée à la boucle décide si l'exécution de cette boucle est bornée ou non.

Nous supposons donc qu'il nous est fourni, en plus de l'algorithme calculant la limite des contenus de la mémoire, un moyen de calculer cette itérabilité. C'est l'algorithme $post^{*,f}$. Nous le détaillons dans chacun des paragrpahes suivants traitant d'une donnée particulière telle que compteurs (section 5.2) ou horloges (section 5.3).

Si l'algorithme d'itérabilité pour une boucle θ donnée, fournit la réponse $faux$, nous ajoutons alors dans l'ensemble $\mathcal{I}_{\mathcal{B}g}$ l'ensemble des transitions apparaissant dans la boucle. En effet, si la boucle n'est pas itérable il ne doit pas être possible de trouver des chemins contenant infiniment souvent des transitions de cette boucle sans avoir pris de temps en temps d'autres transitions.

5.2 Vérification dans le cadre des systèmes à compteurs paramétrés

Dans le cadre des systèmes à compteurs paramétrés nous avons représenté les ensembles de valuations des variables (compteurs) et des paramètres par des matrices de bornes paramétrées contraintes (cf. définition 28, page 55). Les états de contrôle composant le graphe symbolique obtenu lors de l'analyse d'un système à compteurs sont composés d'un état de contrôle $q \in Q$ du modèle initial et d'une matrice contrainte $\tilde{M}$. Nous cherchons ici à caractériser l'itérabilité des composantes strictement connexes contenues dans le graphe symbolique. Pour cela, puisque $\tilde{M}$ représente un ensemble de valuations, il nous faut montrer que pour tous les points contenus dans $\tilde{M}$, toute exécution possible du circuit considéré (caractérisé par la composante connexe) est finie. Nous montrons que sous certaines conditions (conditions de déterminisme) nous sommes

capables de caractériser l'itérabilité de certaines composantes connexes.

Avant de commencer nous allons définir une relation d'ordre sur les composantes connexes contenues dans le graphe pour ensuite analyser "inductivement" ces composantes en commençant par la plus petite selon l'ordre défini. Nous définissons sur le graphe symbolique la notion de *degré de complexité* tel que les circuits simples (élémentaires) sont de degré 1. Le degré des composantes non élémentaires C est défini par induction tel que $degre(C) = max(degre(C')) + 1$ pour toutes les sous-composantes connexes C' de C. Il nous reste un petit détail à définir. En effet, observons la composante strictement connexe représentée par la figure 5.1 (a). Cette composante contient à priori 5 circuits élémentaires, comment faire pour les ordonner. Nous pourrions analyser toutes les combinaisons possibles mais ce serait trop complexe. Nous avons choisi, comme heuristique, de suivre l'ordre induit par le parcours du graphe symbolique. Ainsi, les figures 5.1 (b) et (c) représentent les degrés de complexité que nous avons associés aux composantes selon l'ordre induit par le graphe symbolique. Nous supposons ici que lors du parcours du graphe symbolique, nous entrons dans la composante par l'état sous la flèche incidente dans nos dessins. La composante entière est alors de degré 3 pour le cas (b) et de degré 4 pour le cas (c).

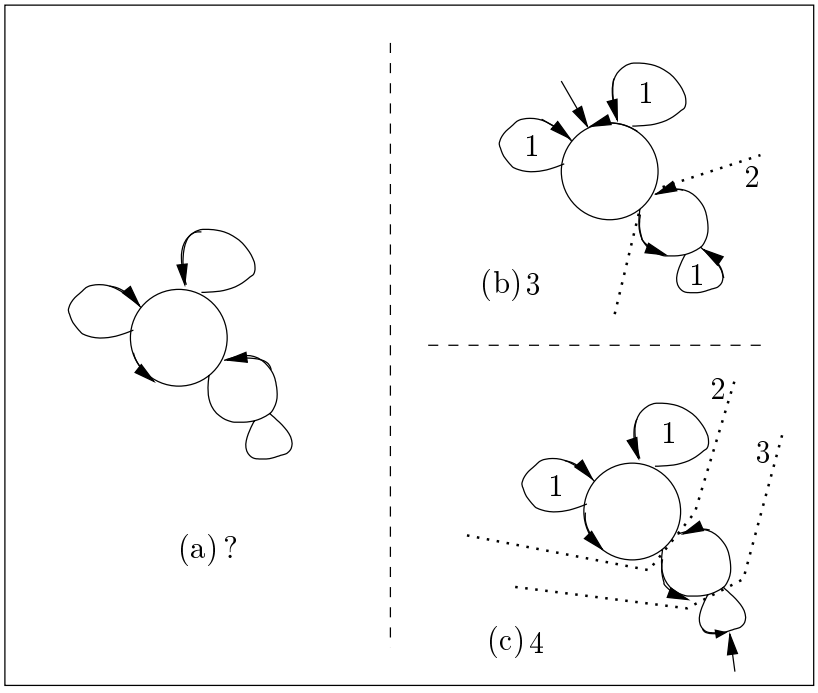


FIG. 5.1 – Ordonnancement des composantes

Nous commençons donc par traiter le cas des circuits simples (de degré de complexité 1) puis nous voyons les cas des composantes complexes.

5.2.1 Circuits simples

Nous rappelons tout d'abord la notion de circuit simple (ou élémentaire).

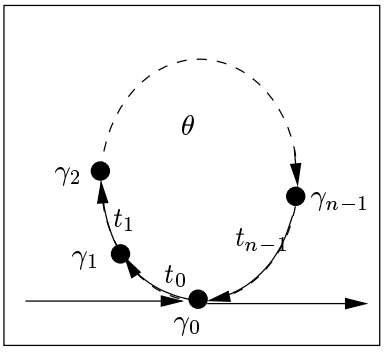


FIG. 5.2 – Circuit simple

Soit $\theta = t_0 t_1 \cdots t_{n-1}$ un circuit tel que $t_i = (q_i, l_i, \tilde{M}_{gi}, (A_i, \overrightarrow{b}_i), q_{i+1})$ pour tout $i \in [0, n-2]$ et $t_{n-1} = (q_{n-1}, l_{n-1}, \tilde{M}_{gn-1}, (A_{n-1}, \overrightarrow{b}_{n-1}), q_0)$. La séquence d'exécution symbolique associée à ce circuit est $\sigma(\theta) = \gamma_0 t_0 \gamma_1 t_1 \cdots t_{n-1} \gamma_0$ où $\gamma_i = (q_i, \tilde{M}_i)$ pour tout $i \in [0, n-1]$. Ce circuit est dit élémentaire si pour tout $i, j \in [1, n-1]$, $i \neq j \Rightarrow q_i \neq q_j$.

5.2.1.1 Effet déterministe

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op_g, Op_v, T)$ un système à compteurs paramétrés. Soit θ un circuit élémentaire de ce système.

Nous montrons maintenant en utilisant les propriétés de Mat^A et Mat^B que l'effet d'une séquence de transitions peut être représenté par un couple (A, B).

Proposition 5

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op_g, Op_v, T)$ un système à compteurs paramétrés. Soit $\sigma = \gamma_0 t_1 \gamma_1 \cdots t_n \gamma_n$ une séquence d'exécution symbolique de ce système telle que : pour tout $i \in [0..n]$, $\gamma_i = (q_i, \tilde{M}_i)$ où $q_i \in Q$ et $\tilde{M}_i \in \tilde{\mathcal{M}}$ et pour tout $j \in [1..n]$, $t_j = (q_{j-1}, l_j, Ec, (A_j, B_j), q_j)$. Nous avons alors $\tilde{M}_n = (A * M_0 * A^T + B, \phi_0)$ telle que

$$\begin{aligned} A &= A_n * \cdots * A_1 \in Mat^A \\ B &= \Sigma_{i=1}^{n} (A_n * \cdots * A_{i+1}) * B_i * (A_n * \cdots * A_{i+1})^T + B_n \in Mat^B \end{aligned}$$

Preuve.

La preuve se fait par induction sur la longueur de la séquence en utilisant les propriétés de Mat^A et Mat^B.

■

Par la proposition 5, nous savons qu'il existe une matrice $A_\theta \in Mat^A$ et une matrice $B_\theta \in Mat^B$ représentant l'effet de l'application de toutes les transitions du circuit. Soit $\gamma_0 = (q_0, \tilde{M}_0)$ une configuration atteignable dans l'état q_0. Nous pouvons alors appliquer le circuit sur cette configuration (sans tenir compte des gardes) nous avons alors $\theta(\tilde{M}_0) = (A_\theta * \tilde{M}_0 * A_\theta^T + B_\theta)$. La proposition suivante exprime le fait que, sous certaines conditions, nous pouvons calculer l'effet de n application(s) de la boucle sur $\tilde{M}_0$, que nous notons $\theta^n(\tilde{M}_0)$, en utilisant les matrices $\tilde{M}_0, A_\theta, B_\theta$. Nous montrons alors que l'ensemble des points calculés de cette façon est l'ensemble des points atteignables.

Proposition 6

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op_g, Op_v, T)$ un système à compteurs paramétrés. Soient $\theta = t_0 t_2 \cdots t_{n-1}$ un circuit simple de ce système, $\gamma_0 = (q_0, \tilde{M}_0)$ une configuration atteignable en q_0 et l'effet du circuit est représenté par (A_θ, B_θ).

Si la matrice distance $\Delta = (A_\theta * \tilde{M}_0 * A_\theta^T + B_\theta) -. \tilde{M}_0$ *satisfait les conditions suivantes :*

$$\begin{array}{lcl} SC1 & : & \forall \mathcal{P}\ \forall n \in \mathcal{N}.\ \phi_1 \Rightarrow (A_\theta * \Delta * A_\theta^T) = \Delta \\ SC2 & : & \Delta \textit{ représente une configuration paramétrée fermée,} \\ & & \textit{c'est-à-dire } \Delta \in Mat^B \textit{ et est fermée} \end{array}$$

alors,

$$\forall \overrightarrow{v} \in [\![\tilde{M}_1]\!]\ \forall n \geq 1.\ \theta^n(\overrightarrow{v}) = (\overrightarrow{d} + n * vecteur(\Delta), \overrightarrow{p})$$

Preuve. La preuve se fait par induction sur le nombre d'itération.

- $(n = 1)$ Appliquer une fois le circuit θ (en ne tenant pas compte des gardes) sur la matrice $\tilde{M}_1$ donne $\theta(\tilde{M}_1) = A_\theta * \tilde{M}_1 * A_\theta^T + B_\theta -. \tilde{M}_1 + \tilde{M}_1 = \tilde{M}_1 + \Delta$. Puisque Δ représente une configuration paramétrée, $\delta = vecteur(\Delta)$ (condition $SC2$) est la distance en terme de point appliquée sur les variables. Ainsi, si $\overrightarrow{\mathrm{v}} \in [\![\tilde{M}_1]\!]$ alors $\theta(\overrightarrow{\mathrm{v}}) = (\overrightarrow{\mathrm{d}} + \delta, \overrightarrow{\mathrm{p}}) \in [\![\theta(\tilde{M}_1)]\!]$.
- $(n \Rightarrow n+1)$ Supposons que $\theta^n(\overrightarrow{\mathrm{v}}) = (\overrightarrow{\mathrm{d}} + n * \delta, \overrightarrow{\mathrm{p}})$ pour tout $\overrightarrow{\mathrm{v}} = (\overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}) \in [\![\tilde{M}_1]\!]$ et $\theta^n(\overrightarrow{\mathrm{v}}) \in [\![\theta^n(\tilde{M}_1)]\!]$. Nous pouvons transposer cette égalite en terme de matrice contrainte ce qui nous donne : $\theta^n(\tilde{M}_1) = \tilde{M}_1^+ + n * \Delta$. Alors, en appliquant $SC1$, nous obtenons :

$$\begin{array}{rcl} \theta^{n+1}(\tilde{M}_1) & = & A_\theta * \theta^n(\tilde{M}_1) * A_\theta^T + B_\theta \\ & = & A_\theta * \tilde{M}_1 * A_\theta^T + A_\theta * n * \Delta * A_\theta^T + B_\theta \\ & = & \tilde{M}_1 + n * (A_\theta * \Delta * A_\theta^T) + \underbrace{A_\theta * \tilde{M}_1 * A_\theta^T + B_\theta -. \tilde{M}_1}_{\Delta} \\ & = & \tilde{M}_1 + (n+1) * \Delta \end{array}$$

  Et enfin puisque Δ représente une configuration paramétrée, nous obtenons bien :

$$\theta^{n+1}(\overrightarrow{\mathrm{v}}) = (\overrightarrow{\mathrm{d}} + (n+1) * \delta, \overrightarrow{\mathrm{p}})$$

■

Remarque : Les deux conditions $SC1$ et $SC2$ correpondent à des conjonctions d'égalité entre des expressions arithmétiques sur les entiers. Ces formules sont donc décidables dans le cas linéaire. Il est donc important de pouvoir simplifier les termes apparaissant dans les matrices contraintes pour obtenir aussi souvent que possible des formules linéaires. Nous rediscutons de ce point dans le chapitre sur l'outil (Chapitre 7).

La condition $SC1$ est similaire à la condition $C1$ donnée par le principe d'extrapolation. La condition $SC2$ exprime le fait que le circuit a un effet *déterministe* à chaque itération puisque la distance entre les configurations symboliques doit être un point (et pas un ensemble) et que l'effet ne dépend pas des variables d'itération (le contraire pourrait apporter des valeurs non déterministes).

5.2.1.2 Itérabilité

Nous venons de calculer l'effet du circuit sans tenir compte des gardes. Caractérisons maintenant la condition d'itérabilité sur les circuits élémentaires.

Intuitivement nous dirons qu'un circuit élémentaire $\theta = t_0 t_1 \cdots t_{k-1}$ est *borné* pour la configuration symbolique $\gamma_0 = (q_0, \tilde{M}_0)$ s'il existe un entier $n \geq 0$ à partir duquel il ne soit plus possible d'itérer le circuit, c'est-à-dire $\theta^n(\gamma_0) = \emptyset$.

Par la proposition 6, nous savons que pour chaque valuation $(\overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}}) \in [\![\tilde{M}_0]\!]$, nous avons $\theta^n(\overrightarrow{\mathrm{v}}) = (\overrightarrow{\mathrm{d}} + n * vecteur(\Delta_\theta), \overrightarrow{\mathrm{p}})$ où Δ_θ est la distance ajoutée aux compteurs. Notons par $t_{[0,i-1]}$ la séquence de transitions $t_0 t_1 \cdots t_{i-1}$ pour tout $i \in [0, k-1]$ et par $t_{[0,i-1]}(\overrightarrow{\mathrm{v}})$ l'effet de la séquence de transitions $t_0 t_1 \cdots t_{i-1}$ sur la valuation des compteurs et des paramètres (si $i = 0$, nous considérons qu'aucune transition n'a été appliquée).

Nous avons alors la proposition suivante :

Proposition 7

Soit $\mathcal{A} = (Q, \mathcal{X}, \mathcal{P}, L, Op_g, Op_v, T)$ un automate étendu et $\mathcal{S}(\mathcal{A})$ le graphe symbolique qui lui est associé. Soit θ un circuit élémentaire du graphe symbolique tel que son effet est Δ_θ. Δ_θ satisfait les conditions $SC1$ et $SC2$. Le circuit θ est **borné** *si et seulement si*

$$\forall \overrightarrow{v} = (\overrightarrow{d}, \overrightarrow{p}) \in [\![\tilde{M}_0]\!] \; \exists n \geq 0. \; \exists i \in [1, k-1].$$

$$(t_{[0,i-1]}(\overrightarrow{d} + n * vecteur(\Delta_\theta), \overrightarrow{p}) \not\models garde(t_i)$$

Intuitivement, la proposition ci-dessus exprime que le circuit du graphe θ est borné si après n itérations, lors de la $n+1$-ième itération une des gardes des transitions n'est plus satifaisable et cela pour tous les points contenus dans la configuration atteignable. Il ne doit, en effet, pas être possible pour certains points de continuer et d'autres de s'arrêter.

Comme la formule de la proposition 7 doit être vraie pour tous les points de départ nous pouvons l'exprimer en utilisant la matrice de bornes paramétrées contrainte initiale $\tilde{M}_0$ et les matrices représentant les gardes $\tilde{M}_{gi}$ (matrice associée à la transition t_i). Nous rappelons que $\phi_0(p, n)$ est la contrainte sur les paramètres associée à la matrice $\tilde{M}_0$.

Nous avons alors :

$$\phi_{it} \equiv \begin{array}{c} \forall p \in \mathcal{P}. \; \forall n \in \mathcal{N}. \; \phi_0(p, n) \\ \Rightarrow \\ \exists m \geq 0. \; m \geq 0 \; \wedge \; \bigvee_{i=0}^{k-1} (\forall c \in \mathcal{X}. \neg (t_{[0i-1]}(\tilde{M}_0^+ + m * \Delta_\theta) \; \wedge \; M_{gi}(p, n, c))) \end{array}$$

où $\tilde{M}_0(p, n, c)$ est la formule sur les compteurs, les paramètres et les variables d'itération que représente la matrice contrainte. La formule ϕ_{it} est une formule décidable dans le cas linéaire (ce qui est le cas si tous les termes des différentes matrices sont linéaires). Si la formule est non linéaire, nous ne pouvons alors donner de réponse. Ainsi, après élimination de toutes les variables, si ϕ_{it} est valide, nous pouvons alors synthétiser une condition d'itérabilité bornée dans le graphe symbolique équitable (dont la définition est donnée page 87) : soit $T(\theta) = \{t_0, \cdots, t_{k-1}\}$ l'ensemble des transitions du circuit θ, nous posons alors :

$$T(\theta) \in I_{\mathcal{B}_g}$$

A partir de la formule de la proposition 7 nous pouvons construire une formule exprimant les conditions sous lesquelles, après n itérations du circuit θ sur un point contenu

dans la configuration $\tilde{M}$ l'exécution s'arrête à cause de la garde de la transition t_i :

$$\phi_i^{\theta,\tilde{M}_0}(\overrightarrow{\mathrm{v}}, n) \equiv t_{[0,i-1]}(\overrightarrow{\mathrm{d}} + n * vecteur(\Delta_\theta), \overrightarrow{\mathrm{p}}) \not\models o_{gi}$$

5.2.2 Circuits complexes

Nous avons deux sortes de circuits non - simples. Dans le cadre d'une composante complexe nous définissons la notion de circuit élémentaire. C'est un circuit simple qui contient la configuration d'entrée de la composante. Il est possible que la composante ne contienne qu'un circuit élémentaire ou bien plusieurs.

Nous étudions tout d'abord les composantes qui ne contiennent qu'un circuit élémentaire, puis nous étendons notre résultat aux composantes contenant plusieurs circuits élémentaires sur l'état d'entrée de cette composante.

Nous donnons tout d'abord dans chacun des cas les conditions de déterminisme puis nous caractérisons la notion d'itérabilité.

Nous supposons que pour toutes sous-composantes connexes C' de la composante C en cours d'analyse nous avons une formule $\phi_i^{C',\tilde{M}_0}(\overrightarrow{\mathrm{v}}, n)$ exprimant le fait que l'exécution de la sous composante C' stoppe dans la configuration γ_i pour la valuation $\overrightarrow{\mathrm{v}}$ après n tours.

5.2.2.1 Cas 1 : 1 seul circuit élémentaire — effet déterministe

Nous nous plaçons tout d'abord dans le cas de composante connexe C n'ayant qu'un seul circuit simple θ qui contienne la configuration d'entrée $\gamma_0 = (q_0, \tilde{M}_0)$: les sous composantes connexes de C, différentes de θ peuvent partagées zéro ou plus de configurations symboliques avec θ (cf. la première partie de la figure 5.4).

Puisque nous venons d'analyser le cas simple, nous allons effectuer des transformations sur la structure de la composante pour se ramener au cas des circuits simples. La méthode utilisée pour cela se décompose en deux parties.

- La première partie consiste à développer les circuits des sous-composantes connexes qui ont plus d'une configuration symbolique en commun avec celle de la séquence d'exécution induite par θ pour qu'elles ne soient plus attachées au circuit élémentaire que par une seule configuration.
- La seconde est de transformer chacune de ces sous-composantes (plus ou moins complexes) en méta-transitions. Ces méta-transitions sont des transitions que nous allons ajouter à la composante à la place des sous-composantes. Elles représentent l'effet de ces sous-composantes.

Ainsi, nous commençons par construire, à partir de la composante C, une composante C_θ telle que chaque sous-composante de C_θ ne soit reliée que par une seule configuration symbolique au circuit élémentaire. Nous construisons C_θ de telle sorte qu'elle ait le même degré de complexité et les mêmes séquences d'exécutions que C. En regardant la figure 5.3, nous nous apercevons qu'il existe plusieurs transformations possibles : à partir d'une sous-composante partageant 3 configurations avec θ il est possible d'obtenir 3 décompositions vérifiant les caractéristiques que nous venons d'énoncer.

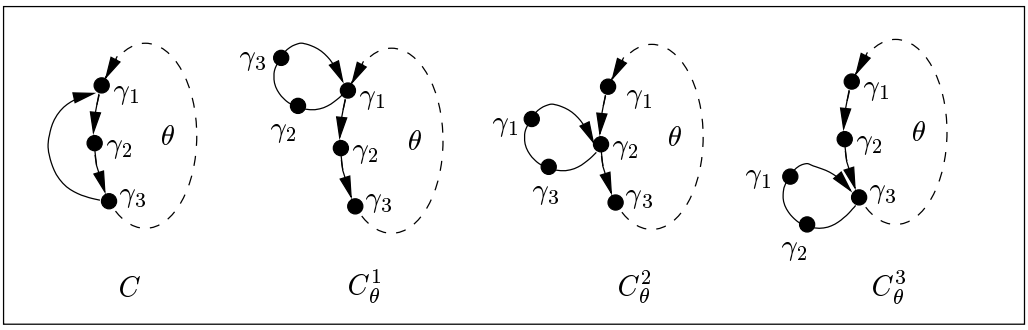


FIG. 5.3 – Différentes transformations possibles

De manière à obtenir toujours la même transformation, nous ordonnons les configurations symboliques contenues dans θ en suivant la relation de transition, nous avons ainsi $\gamma_0 \prec \gamma_1 \prec \cdots \prec \gamma_{k-1}$. Puisque θ est l'unique circuit élémentaire de C, l'unique chemin reliant deux configurations γ_i, γ_j ($i < j$) est contenu dans θ. Soit γ_i une configuration partagée par θ et une des sous-composantes de C. Nous notons par $CC(\gamma_i)$ la sous-composante connexe maximale de C n'incluant pas θ, telle que γ_i est la configuration minimale (au sens de $\prec$) des configurations de $CC(\gamma_i)$ partagées par θ.

Alors, C_θ est construite à partir de θ telle que pour chaque configuration γ_i de θ, nous ajoutons une copie de $CC(\gamma_i)$ pour que la seule configuration partagée par $CC(\gamma_i)$ et θ soit γ_i. Dans l'exemple de la figure 5.3 c'est la décomposition C^1_θ qui est retenue.

Un exemple plus complexe de la transformation est donnée par la figure 5.4.

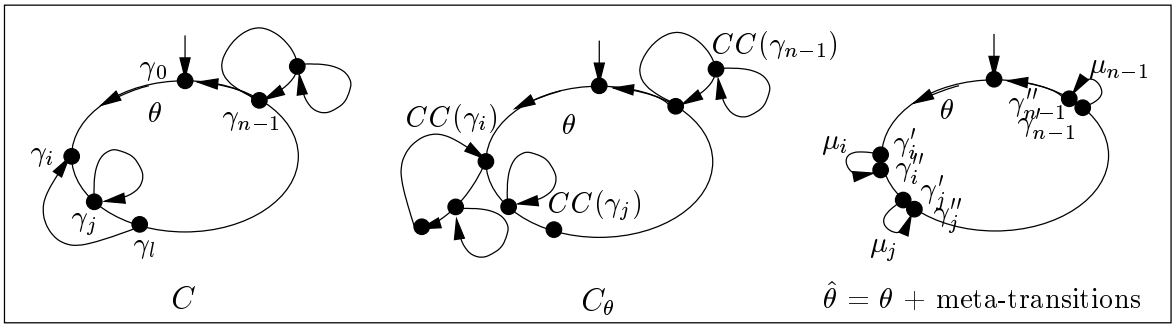


FIG. 5.4 – Construire $\hat{\theta}$ de C

Lemme 6

Les chemins d'exécutions de C et de C_θ sont identiques.

Preuve. La preuve se fait par construction de C_θ.

■

Observons l'effet d'une itération de la composante sur un point $\overrightarrow{\mathrm{v}} = (\overrightarrow{\mathrm{d}}, \overrightarrow{\mathrm{p}})$ (valuation des compteurs et des paramètres) :

$$C(\overrightarrow{\mathrm{v}}) = \{\overrightarrow{\mathrm{v}}' \mid \exists n_1 \cdots n_{k-1} \geq 0.\ \overrightarrow{\mathrm{v}}' = (t_{k-1} \circ CC(\gamma_{k-1})^{n_{k-1}} \circ \cdots \circ CC(\gamma_1)^{n_1} \circ t_0)(\overrightarrow{\mathrm{v}})\}$$

où $t_i(\overrightarrow{\nu})$ est l'application de la transition t_i et $CC(\gamma_i)^{n_i}(\overrightarrow{\nu})$ est l'effet de n_i itérations de la sous-composante dans l'état γ_i sur la valuation $\overrightarrow{\nu}$ (nous supposons que nous avons son effet puisque la sous-composante est de degré inférieur à celui de la composante C).

La cardinalité de cet ensemble est supérieure ou égale à 1, puisque les variables d'itération n_i peuvent introduire des comportements non-déterministes : il peut exister plusieurs valeurs de n_i pour lesquelles la composante C peut être appliquée.

Si l'effet de la composante donne plus d'une valuation nous ne pourrons pas représenter l'effet pour n itération(s). En fait, pour être capable de calculer cet effet, nous allons imposer des conditions supplémentaires telles que l'application de C soit déterministe, c'est-à-dire donne toujours le même resultat.

Passons maintenant à la deuxième partie de notre méthode : la transformation en boucle simple. Supposons qu'il soit possible pour chaque sous-composante connexe $CC(\gamma_i)$ de calculer son effet et que cet effet soit représentable par une distance $\Delta_{CC(\gamma_i)}$ satisfaisant les conditions de la proposition 6. Nous allons alors transformer la composante C_θ pour nous ramener au cas d'un circuit simple. Nous pouvons en effet remplacer chaque sous-composante $CC(\gamma_i)$ par une méta-transition $t^n_{CC(\gamma_i)}$ telle que son opération soit $x' := x + n * vecteur(\Delta_{CC(\gamma_i)})$ où $n \geq 0$ est une nouvelle variable d'itération (nous choisissons une variable inutilisée pour chaque méta-transition). Nous remarquons que les éléments du vecteur $n * vecteur(\Delta_{CC(\gamma_i)})$ sont bien des termes arithmétiques sur $\mathcal{P} \cup \mathcal{N}$.

Cette méta-transition $t^n_{CC(\gamma_i)}$ est introduite dans le graphe en remplaçant les sous-composantes $CC(\gamma_i)$. Il faut pour cela dédoubler les configurations γ_i en γ'_i et γ''_i telles que (cf. figure 5.4) :

- γ'_i est la configuration source de la méta-transition et la configuration arrivée de la transition entrante de γ_i dans la composante C_θ.
- γ''_i est la configuration arrivée de la méta-transition et la configuration source de la transition sortante de γ_i dans la composante C_θ.

Ainsi, nous obtenons un circuit élémentaire, appelé $\hat{\theta}$, construit à partir des transitions de θ et des méta-transitions mettant en œuvre des variables d'itération. Le résultat de la proposition 6 peut alors s'appliquer à $\hat{\theta}$ ce qui nous donne :

Proposition 8
Si $\Delta_{\hat{\theta}} = A_{\hat{\theta}} * \tilde{M}_0 * A^T_{\hat{\theta}} + B_{\hat{\theta}} -$. $\tilde{M}_0$ *satisfait les conditions SC*1 *et SC*2 *alors*

$$\forall \overrightarrow{v} = (\overrightarrow{d}, \overrightarrow{p}) \in [\![\tilde{M}_0]\!] \forall n \geq 0.\ C^n(\overrightarrow{v}) = (\overrightarrow{d} + n * vecteur(\Delta_{\hat{\theta}}), \overrightarrow{p})$$

Nous venons de caractériser l'effet d'une composante complexe sur les points.

Nous rappelons que la condition $SC2$ revient à demander à ce que $\Delta_{\hat{\theta}}$ représente un point et ne contienne pas de variables d'itération, il en résulte que l'ensemble $C(\overrightarrow{\nu})$ est de cardinalité 1. C'est possible lorsque le circuit θ contient des transitions permettant de faire une "remise à zéro" des valeurs des compteurs en permettant ainsi d'annuler les effets des variables d'itération. Une "remise à zéro" dans le cadre des systèmes à compteurs est soit une transition ayant pour opération sur tous les compteurs $c_i := b_i$ (tous les compteurs sont réinitialisés), soit tous les compteurs sont testés dans une garde avec une contrainte de la forme $c_i = t_i$ (pour passer cette transition la valeur des compteurs doit être égale à t_i).

Remarque : Dans certains cas le circuit θ ne contenant pas de "remise à zéro", nous devons alors traiter la présence de variable d'itération dans $\Delta_{\hat{\theta}}$.

Par exemple lorsque toutes les opérations de C accumulent les valeurs des compteurs, i.e. toutes les assignations sont de la forme $x := x + t$, les (méta)-transitions de $\hat{\theta}$ accumulent leur résultat et l'effet du circuit peut être donné par :

$$C^n(x) = x + \Sigma_{1 \leq i < \infty} n_i * vecteur(\Delta_i)$$

où $\Delta_i \in Mat^B$ sont construits en terme d'assignations des méta-transitions.

En fait, dans le cas général nous ne pouvons pas calculer l'effet du circuit. Le cas particulier des transitions déterministes nous permet tout de même d'analyser une large classe de système.

5.2.2.2 Cas 1 : 1 seul circuit élémentaire — Itérabilité

Nous rappelons que le circuit élémentaire de la composante connexe que nous analysons est composé de k transitions. Nous notons par $garde(t_i)$ la formule représentant la garde de la transition t_i.

Nous allons construire une formule $\phi_j^{\hat{\theta},\gamma_0}(\overrightarrow{\mathrm{v}}, n)$ exprimant le fait qu'après n itérations de $\hat{\theta}$ sur la configuration réelle $\overrightarrow{\mathrm{v}} \in [\![\gamma_0]\!]$, l'exécution de $\hat{\theta}$ est arrêtée à la $j$ème transition $(0 \leq j \leq k-1)$. Cette formule est la disjonction de trois sous formules exprimant tous les cas d'arrêts possibles :

1. C'est la transition sortante de γ_j qui nous bloque. Nous sommes bloqués par le circuit élémentaire.

   $$\phi_{j,(1)}^{\hat{\theta},\gamma_0}(\overrightarrow{\mathrm{v}}, n) = \exists m_0, \cdots, m_j \geq 0.\ \hat{\theta}[0, j-1]((\overrightarrow{\mathrm{d}} + n * vecteur(\Delta_{\hat{\theta}}), \overrightarrow{\mathrm{p}})) \not\models garde(t_j)$$

   Les variables entières $m_0, \cdots, m_j$ sont les variables d'itérations utilisées dans les méta-transitions de la séquence $\hat{\theta}[0, j-1]$. Nous pouvons les quantifier existanciellement car nous sommes dans le cas déterministe.

2. L'exécution stoppe dans la composante $CC(\gamma_j)$, mais ce n'est pas la première transition de cette composante qui stoppe. En effet, si l'exécution s'arrête à cause de la première transition cela ne veut rien dire au sujet de l'itérabilité, il est peut être possible de continuer sur le circuit élémentaire.

   $$\phi_{j,(2)}^{\hat{\theta},\gamma_0}(\overrightarrow{\mathrm{v}}, n) = \begin{array}{c} \exists m_0, \cdots, m_j, m \geq 0. \\ \bigvee_{l \neq 0} \phi_l^{CC(\gamma_j), \tilde{M}_j}(\hat{\theta}[0, j-1](\overrightarrow{\mathrm{d}} + n * vecteur(\Delta_{\hat{\theta}}), \overrightarrow{\mathrm{p}}), m) \end{array}$$

   La variable l parcours l'ensemble des configurations du circuit élémentaire de la composante $CC(\gamma_j)$.

3. L'exécution de la composante $CC(\gamma_j)$ ne termine pas.

   $$\phi_{j,(3)}^{\hat{\theta},\gamma_0}(\overrightarrow{\mathrm{v}}, n) = \begin{array}{c} \exists m_0, \cdots, m_j \geq 0.\ \forall m \geq 0. \\ \neg\phi^{CC(\gamma_j), \tilde{M}_j}(\hat{\theta}[0, j-1](\overrightarrow{\mathrm{d}} + n * vecteur(\Delta_{\hat{\theta}}), \overrightarrow{\mathrm{p}}), m) \end{array}$$

   Dans ce cas, l'hypothèse selon laquelle *toutes* les transitions de C doivent être infiniment souvent exécutées n'est pas satisfaite.

Ainsi, si C est une composante complexe ne contenant qu'un circuit élémentaire nous avons la proposition suivante :

Proposition 9

Soit C une composante de degré de compléxité supérieur à 1 ne contenant qu'un seul circuit élémentaire. Soit $\gamma_0 = (q_0, \tilde{M}_0)$ la configuration d'entrée de cette composante. L'effet de cette composante est Δ_C. Alors l'exécution de C est bornée si et seulement si

$$\forall P \forall N \forall \overrightarrow{v} \in [\![\tilde{M}_0]\!].\ \tilde{M}_0(P,N)[\overrightarrow{v}/\mathcal{X},\mathcal{P}] \Rightarrow \exists n \geq 0.\ \bigvee_{j \in [0,k-1]} \phi_j^{\hat{\theta},\tilde{M}_0}(\overrightarrow{v}, n)$$

Si la composante est bornée nous pouvons alors ajouter l'ensemble composé de toutes les transitions de C dans l'ensemble $\mathcal{I}_{\mathcal{B}_g}$.

Cette formule est décidable dans le cadre des formules linéaires. Nous pouvons transformer cette formule sur les points en une formule conservant les matrices grâce au déterminisme.

5.2.2.3 Cas 2 : plusieurs circuits élémentaires — effet déterministe

Considérons maintenant les composantes connexes C ayant plusieurs circuits élémentaires contenant la configuration d'entrée $\gamma_0 = (q_0, \tilde{M}_0)$. Nous notons par $\Theta(C)$ l'ensemble de ces circuits élémentaires. Par exemple, l'ensemble $\Theta(C)$ de la composante connexe C représentée par le dessin de gauche de la figure 5.5 est égal à $\{\theta_1 = (t_0 t_1 t_3),\ \theta_2 = (t_0, t_1, t_4),\ \theta_3 = (t_5, t_6, t_7, t_3)\}$.

Pour analyser l'effet de C nous voulons construire une composante connexe $\tilde{C}$ ayant le même degré de complexité et les mêmes séquences d'exécution que C, mais où chaque sous-composante connexe contenant γ_0 ne contient qu'un seul circuit élémentaire de $\Theta(C)$. De plus, les sous-composantes prises deux à deux ne partagent que γ_0. Ainsi chaque sous-composante prise séparément entre dans le cas explicité précédemment (composante avec un seul circuit connexe). Nous en connaissons donc l'effet. Ayant l'effet de chaque sous-composante, nous allons calculer l'effet de la composante totale C.

Nous avons représenté dans la figure 5.5 un exemple d'une telle décomposition.

Voyons maintenant comment construire effectivement $\tilde{C}$. Considérons le cas de θ_i un élément de $\Theta(C)$. A partir de θ_i, nous extrayons la sous-composante connexe maximale, notée $CC(\theta_i)$, contenant les états et les transitions de θ_i plus les états et les transitions de C qui n'appartiennent pas à un autre circuit élémentaire de $\Theta(C)$. $EC(C)$ représente l'ensemble de toutes ces sous-composantes :

$$EC(C) = \{CC(\theta_i) | \theta_i \in \Theta(C)\}$$

Enfin, $\tilde{C}$ est construit à partir de $EC(C)$ en dupliquant les configurations tel que le seul état partagé par chaque pair de sous-composante $CC(\theta_i)$ soit la configuration initiale γ_0.

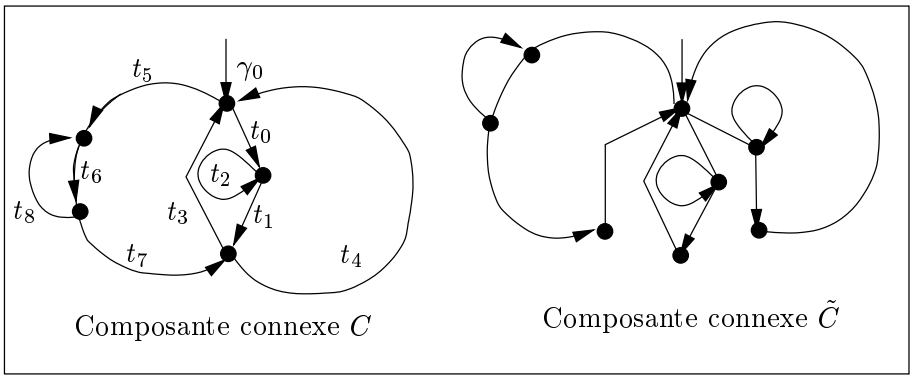


FIG. 5.5 – Construction de $\tilde{C}$

Lemme 7

Soit C une composante connexe du graphe symbolique contenant plusieurs circuits élémentaires. Soit $\tilde{C}$ la composante dérivée de C construite comme expliqué ci-dessus. Nous avons alors le résultat suivant :

C et $\tilde{C}$ ont le même ensemble de séquence d'exécution.

L'effet d'une itération de C sur une valuation $\overrightarrow{\nu}$ est

$$C(\overrightarrow{\nu}) = \bigcup_{\theta_i \in \Theta(C)} (\bigcup_{j \neq i} CC(\theta_j))^* CC(\theta_i)(\overrightarrow{\nu})$$

Intuitivement, l'effet d'une itération de C revient à exprimer toutes les exécutions possibles des différents circuits élémentaires : il est possible pour chaque circuit θ_i que nous ayons tourné n fois ($n \geq 0$) dans les autres circuits puis que nous passions une fois dans θ_i.

Donc, si nous considérons séparément chaque sous-composante $CC(\theta_i)$ de $\tilde{C}$, nous pouvons appliquer la proposition 8 pour calculer son effet. Et ainsi nous pouvons réduire la sous-composante à une méta-transition représentant son effet (si elle satisfait les conditions de la proposition). Qui dit méta-transition, dit insertion d'une variable d'itération n différente pour chaque circuit élémentaire. Nous devons définir les conditions sous-lesquelles il nous est possible de caractériser l'effet de la composante C pour vérifier que cet effet est exact.

Tout d'abord, nous devons être capables pour chaque sous-composante de calculer son effet $\Delta_{CC(\theta_i)}$. Cet effet doit satisfaire les conditions $SC1$ et $SC2$.

Ensuite, nous voulons exprimer l'effet de la composante pour chaque circuit θ_i : $(\bigcup_{j \neq i} CC(\theta_j))^* CC(\theta_i)$. Puisque nous avons l'effet pour chaque $\theta \in \Theta$ cela nous donne l'effet suivant :

$$\Delta_{CC^+(\theta_i)} = n_1 * \Delta_{CC(\theta_1)} + n_2 * \Delta_{CC(\theta_2)} + \cdots + n_l * \Delta_{CC(\theta_l)} + \Delta_{CC(\theta_i)}$$

où $\forall j \in [1, l]$, $\theta_j \neq \theta_i$ et n_j sont des variables d'itérations différentes.

$\Delta_{CC^+(\theta_i)}$ n'appartient donc pas à Mat^B. Nous ne pouvons pas exprimer $n * \Delta_{CC^+(\theta_i)}$ dans le cas général.

Par contre, si l'effet des sous-composantes $CC(\theta_i)$ est le même :

$$\Delta_{CC(\theta_1)} = \Delta_{CC(\theta_2)} = \cdots = \Delta_{CC(\theta_n)} = \Delta_C$$

Chaque expression $n * \Delta_{CC^+(\theta_i)}$ est maintenant égale à $n * \Delta_C$. L'effet de la composante C est alors Δ_C.

Proposition 10

Soit C une composante connexe de degré de complexité $degre(C) \geq 1$ et $\gamma_0 = (q_0, \tilde{M}_0)$ la configuration initiale. Si C satisfait les conditions suivantes :

- *Chaque sous-composante stricte de C peut être réduite à une méta-transition.*
- *pour tous les circuits élémentaires θ_i contenant l'état initial de C, son effet est $\Delta_{CC(\theta_i)}$, il satisfait les conditions $SC1$ et $SC2$. Les effets de toutes les composantes $CC(\theta_i)$ sont égaux $\Delta_C \in Mat^B$ et fermés.*

alors,

$$\forall \overrightarrow{v} \in [\![\tilde{M}_0]\!] \forall n \geq 0.\ C^n(\overrightarrow{v}) = (\overrightarrow{d} + n * vecteur(\Delta_C), \overrightarrow{p})$$

5.2.2.4 Cas 2 : plusieurs circuits élémentaires — Itérabilité

Nous montrons comment synthétiser les contraintes d'équité pour les composantes connexes complexes ayant plusieurs circuits élémentaires. Nous prenons par hypothèse que ces composantes satisfont les conditions de la propriété 10.

Pour que les exécutions dans la composante C soient bornées nous devons vérifier que toutes les sous-composantes sont elles-mêmes bornées.

Proposition 11

Soit C une composante complexe contenant plusieurs circuits élémentaires, soient $\Theta(C)$ l'ensemble de ces circuits et $EC(C)$ l'ensemble de toutes les sous-composantes de C, soit γ_0 la configuration d'entrée dans le composante C, si C satisfait les conditions de la proposition 10, C est bornée si la formule suivante est valide :

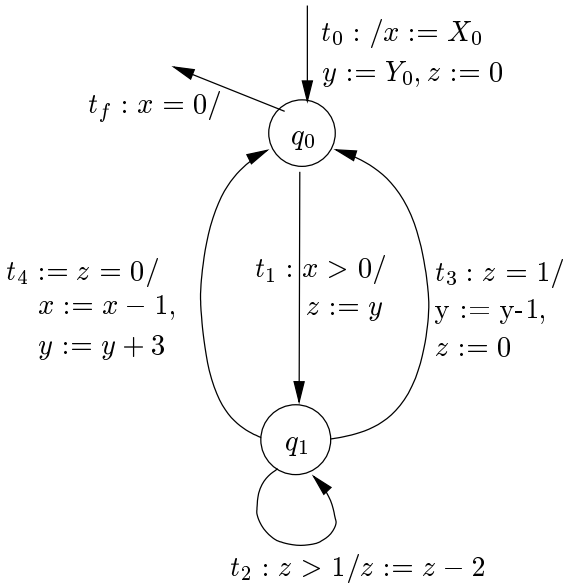
$$\forall P \forall N \forall \overrightarrow{v} \in [\![\gamma_0]\!].\ \tilde{M}_0(P,N)[\overrightarrow{v}/\mathcal{X}, \mathcal{P}] \Rightarrow \bigwedge_{\theta \in \Theta(C)} \exists n \geq 0.\ \phi^{EC(\theta),\gamma_0}(\overrightarrow{v}, n)$$

5.2.3 Un exemple

Pour se convaincre de l'utilité de l'analyse des boucles complexes étudions l'exemple décrit par l'algorithme suivant (cf. figure 5.2.3 (a)).

$x := X_0;\ y := Y_0;\ z := 0;$
tant que $x > 0$ **faire**
  $z := y$
  tant que $z > 1$ **faire**
    $z := z - 2$
  ftq
  si $z = 0$
  alors
    $x := x - 1;$
    $y := y + 3;$
  sinon
    $y := y - 1;$
    $z := 0;$
  fsi
ftq

(a) Algo. de l'exemple



(b) le système à compteurs

FIG. 5.6 – Vérification de propriété : un exemple

Les approches basées sur le calcul de fonctions de mesures [DGG00, CS01] ne peuvent pas prouver que le programme décrit par l'algorithme ci-dessus termine. En fait, les fonctions de mesures synthétisées pour la boucle la plus interne (celle sur z) et pour la boucle la plus externe (celle de x) ne produisent pas l'ordre lexicographique nécessaire à la vérification de la terminaison. C'est bien un ordre lexicographique dont nous avons besoin car le comportement de la boucle interne dépend de la valeur de y qui est calculée par la boucle externe et le comportement de la boucle externe est lui-même dépendant de la valeur de z qui est calculée par la boucle interne. En fait, si nous regardons les premières valeurs des variables nous avons initialement

$$x = X_0 \text{ et } y = Y_0$$

Supposons que la valeur Y_0 soit paire, nous obtenons alors après itération de la boucle externes les valeurs suivantes :

$$x = X_0 - 1 \text{ et } y = Y_0 + 3$$

Nous réappliquons le circuit ce qui donne (la valeur de y est maintenant impaire) :

$$x = X_0 - 1 \text{ et } y = Y_0 + 3 - 1 = Y_0 + 2$$

Nous remarquons que lorsque la valeur de x ne change pas celle de y diminue et lorsque la valeur de x diminue celle de y augmente. La fonction de mesure doit donc être basée sur un ordre lexicographique sur le couple de variables (x, y).

En appliquant l'algorithme de calcul des configurations atteignables, avec la notion d'accélération, nous obtenons le graphe symbolique fini représenté par la figure 5.7, dont nous avons repris les principales configurations dans le tableau 5.1.

Regardons maintenant si nous pouvons vérifier la terminaison de cet algorithme. Pour cela, nous devons observer chacune des composantes connexes du graphe symbolique et statuer sur leur itérabilité. Nous remarquons que le graphe est composé principalement de trois composantes :

- Les composantes simples construites à partir de la transition t_2. Elles sont au nombre de 5 dans le graphe, elles nous permettent de savoir si la valeur de z est paire ou impaire.
- Les composantes commençant en γ'_0 et γ''_0 correspondent à l'application de la boucle externe pour respectivement les valeurs impaires et paires du paramètre Y_0.

En fait, à partir de l'état initial γ_0, nous appliquons la boucle externe. Ainsi, après avoir exécuté la séquence $t_0t_1t_2t_2^*$ (l'itération de t_2 ayant introduit le paramètre n_0), nous obtenons deux cas. Ces deux cas dépendent de la valeur initiale de Y_0, si elle est paire c'est la transition t_4 qui est choisie dans γ_2, si elle est impaire c'est la transition t_3 (ce qui est exprimé par les contraintes ϕ''_0 et ϕ'_0).

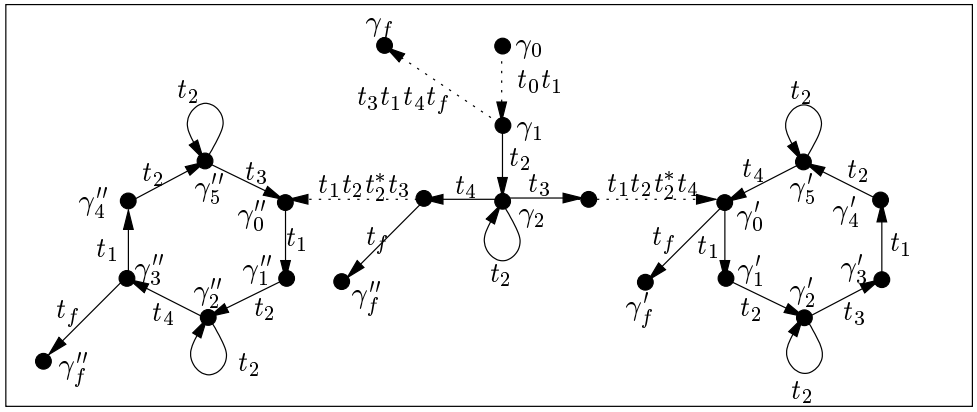


FIG. 5.7 – Graphe symbolique de l'exemple

Etudions tout d'abord l'itérabilité des boucles simples. La distance de cette boucle est $(x = 0,\ y = 0,\ z = -2)$ qui satisfait bien les conditions SC_1 et SC_2. La boucle est alors bornée si et seulement si pour chacune des configurations sur laquelle s'applique cette boucle, la formule d'itérabilité (cf. proposition 7, page 92) est valide. Par exemple, pour la configuration γ_2, nous cherchons la validité de la formule suivante (pour plus de lisibilité nous avons enlevé les contraintes sur x et y car la boucle n'affecte pas ces variables) :

$$\phi_{it}^{\gamma_2} \equiv \begin{array}{c} \forall Y_0\ \forall n_0\ \exists m\ \forall z.(n_0 \geq 0\ \wedge\ Y_0 \geq 0) \\ \Rightarrow \\ m \geq 0\ \wedge\ \neg(z = Y_0 - 2 * n_0 - 2 - 2m\ \wedge\ z > 1) \end{array}$$

TAB. 5.1 – Configurations symboliques principales de la Figure 5.7.

conf.	état contr.	PDBM x	PDBM y	PDBM z	contrainte
γ_0	q_0	X_0	Y_0	0	$\phi_0 \equiv X_0, Y_0 \geq 0$
γ_2	q_0	X_0	Y_0	$Y_0 - 2n_0 - 2$	$\phi_0, Y_0 - 2n_0 > 1, n_0 \geq 0$
γ'_0	q_0	X_0-n'_1-1	Y_0+$2n'_1$+2	0	$\phi'_0 \equiv Y_0 = 2n_0$+3,$X_0, Y_0, n'_1, n_0 \geq 0$
γ'_1	q_1	X_0-n'_1-1	Y_0+$2n'_1$+2	Y_0+$2n'_1$+2	ϕ'_0
γ'_2	q_1	X_0-n'_1-1	Y_0+$2n'_1$+2	Y_0+$2n'_1$-$2n'_2$	ϕ'_0,$n'_2 \geq 0$
γ'_3	q_0	X_0-n'_1-1	Y_0+$2n'_1$+1	0	ϕ'_0
γ'_4	q_1	X_0-n'_1-1	Y_0+$2n'_1$+1	Y_0+$2n'_1$+1	ϕ'_0
γ'_5	q_1	X_0-n'_1-1	Y_0+$2n'_1$+1	Y_0+$2n'_1$-$2n'_3$-1	ϕ'_0,$n'_3 \geq 0$
γ''_0	q_0	X_0-n''_1-1	Y_0+$2n''_1$+2	0	$\phi''_0 \equiv Y_0 = 2n_0$,$X_0, Y_0, n''_1, n_0 \geq 0$
γ''_1	q_1	X_0-n''_1-1	Y_0+$2n''_1$+2	Y_0+$2n'_1$+2	ϕ''_0
γ''_2	q_1	X_0-n''_1-1	Y_0+$2n''_1$+2	Y_0+$2n''_1$-$2n''_2$	ϕ''_0,$n''_2 \geq 0$
γ''_3	q_0	X_0-n''_1-2	Y_0+$2n''_1$+5	0	ϕ''_0
γ''_4	q_1	X_0-n''_1-2	Y_0+$2n''_1$+5	Y_0+$2n''_1$+5	ϕ''_0
γ''_5	q_1	X_0-n''_1-2	Y_0+$2n''_1$+5	Y_0+$2n''_1$-$2n''_3$+ 3	ϕ''_0,$n''_3 \geq 0$

Cette formule est valide, la boucle induite par t_2 sur la configuration γ_2 est donc bornée. Nous pouvons de la même manière vérifier que toutes les composantes de degré 1 de cet exemple sont bornées.

Etudions maintenant l'itérabilité de la composante connexe $t_1 t_2 t_2^* t_3 t_1 t_2 t_2^* t_4$ commençant dans la congfiguration γ'_0. Le vecteur associé à la distance induite par le circuit sur γ'_0 est $(x = -1,\ y = 2,\ z = 0)$. Cette distance satisfait les conditions SC_1 et SC_2. Nous remarquons que l'effet de la boucle interne (introduisant des variables d'itérations) est annulé par les transitions t_3 et t_4. Nous avons ici une composante complexe contenant un seul circuit élémentaire. Vérifier l'itérabilité d'une telle composante c'est vérifier pour chacune des configurations contenues dans le circuit élémentaire que l'exécution s'arrête pour l'une des trois raisons que nous avons énoncées ci-avant (arrêt sur le circuit élémentaire, arrêt dans une sous-composante ou exécution infinie d'une sous-composante).

Il n'y a pas d'itération infinie dans les sous-composantes car ce sont les circuits simples introduits par la transition t_2 que nous venons de montrer bornés. De plus ces circuits simples n'ont qu'une seule transition, nous ne sommes donc pas dans le cas d'un arrêt dans une sous-composante. Finalement, l'exécution de cette composante complexe s'arrête car la garde de la transition t_1 n'est plus satisfaite. La formule le vérifiant est la suivante :

$$\begin{gathered}\forall n'_1\ \forall X_0\ \forall Y_0\ \forall n_0\ \exists n\ \forall x\ \forall y.\\ (X_0 \geq 0\ \wedge\ Y_0 \geq 0\ \wedge\ n'_1 \geq 0\ \wedge\ n_0 \geq 0)\\ \Rightarrow\\ n \geq 0\ \wedge\ \neg(x = X_0 - n'_1 - 1 - n\ \wedge\ y = Y_0 + 2n'_1 + 2 + 2n\ \wedge\ x > 0)\end{gathered}$$

Cette formule est valide, la garde de la transition n'est plus satisfaite à partir d'un nombre de tours donnés.

Nous venons de prouver que l'algorithme se termine pour toutes les valeurs de Y_0 et X_0.

5.3 Vérification dans le cadre des systèmes à horloges et/ou compteurs paramétrés

A partir du moment où nous considérons des systèmes contenant des horloges, il n'est pas possible de considérer l'effet d'un circuit simple (contenant plusieurs transitions) par un couple (A, B). En effet, dans le cadre des horloges, le passage du temps implique la relation suivante :

$$\uparrow(A_2 * \uparrow(A_1 * M * A_1^T) * A_2^T) \neq \uparrow(A_2 * A_1 * M * A_1^T * A_2^T)$$

Pourtant nous pouvons appliquer la théorie décrite pour les compteurs en prenant pour distance référence celle calculée lors de l'analyse du système (cette dernière tient compte des gardes pour les horloges). En effet, lorsque deux distances sont équivalentes dans le cadre des horloges c'est que l'effet du circuit est bien déterministe.

Voici un petit exemple. Le système est composé d'un état de contrôle et de deux horloges x et y. Les transitions sont représentées par la figure 5.8.

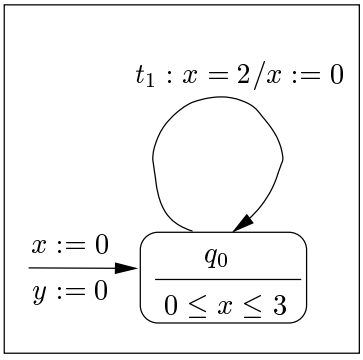


FIG. 5.8 – Un exemple

La matrice initiale avec laquelle nous commençons l'analyse est la suivante :

$$\begin{pmatrix} (\leq,0) & (\leq,0) & (\leq,0) \\ (\leq,3) & (\leq,0) & (\leq,0) \\ (\leq,3) & (\leq,0) & (\leq,0) \end{pmatrix}$$

Si nous appliquons deux fois la transition nous obtenons :

$$\begin{pmatrix} (\leq,0) & (\leq,0) & (\leq,-2) \\ (\leq,3) & (\leq,0) & (\leq,-2) \\ (\leq,5) & (\leq,2) & (\leq,0) \end{pmatrix} \quad \begin{pmatrix} (\leq,0) & (\leq,0) & (\leq,-4) \\ (\leq,3) & (\leq,0) & (\leq,-4) \\ (\leq,7) & (\leq,4) & (\leq,0) \end{pmatrix}$$

Les distances calculées sont :

$$\Delta_0 = \begin{pmatrix} 0 & 0 & -2 \\ 0 & 0 & -2 \\ 2 & 2 & 0 \end{pmatrix} \quad \Delta_1 = \begin{pmatrix} 0 & 0 & -2 \\ 0 & 0 & -2 \\ 2 & 2 & 0 \end{pmatrix}$$

La distance représente bien un point, nous pouvons appliquer la proposition 7 et tester la formule suivante :

$$\forall n\ \exists m\ \forall x\ \forall y.\ (n \geq 0)$$
$$\Rightarrow$$
$$m \geq 0\ \wedge$$
$$\neg(0 \leq x \leq 3\ \wedge\ 2n + 2 + 2m \leq y \leq 5 + 2n + 2m$$
$$\wedge\ x - y = -2n - 2 - 2m\ \wedge\ x = 2)$$

Cette formule n'est pas valide, le circuit n'est donc pas borné. En effet, seuls les points qui dépassent la garde $(x \leq 2)$ s'arrètent, pour les autres ils peuvent continuer.

Maintenant si nous ajoutons la contrainte $y <= 15$ sur la garde nous devons tester la formule :

$$\forall n\ \exists m\ \forall x\ \forall y.\ (n \geq 0)$$
$$\Rightarrow$$
$$m \geq 0\ \wedge\ \neg(0 \leq x \leq 3\ \wedge\ 2n + 2 + 2m \leq y \leq 5 + 2n + 2m$$
$$\wedge\ x - y = -2n - 2 - 2m\ \wedge\ x = 2\ \wedge\ y \leq 15)$$

Cette formule est valide, cette fois ci le circuit est bien borné pour tous les points.

5.4 Discussion

Pour ce qui est de la vérification de propriétés de justice, certains travaux vérifient ces propriétés dans des cadres très particuliers :

- Dans les travaux de [CS01, PS00] les techniques proposées pour la vérification automatique des ces propriétés sont dans le cadre du model-checking régulier, pour le cas particulier des transformations de séquences préservant la longueur.
- Dans [BLS00] la technique employée pour la transposition des conditions de justice du modèle concret au modèle fini abstrait, ainsi que la procédure de calcul des conditions de justice à ajouter au modèle abstrait sont proposées dans le cas particulier des réseaux de processus séquentiels paramétrés.

Dans le cadre de nos travaux, nous ne nous restreignons pas à des classes particulières mais donnons des principes pouvant s'appliquer à différentes classes.

D'autre part, il existe des travaux sur la vérification de propriété de vivacité basés sur l'analyse de la terminaison des programmes [Flo67, MP91]. Ces travaux ne proposent pas de méthodes automatiques, elles sont basées sur le principe de preuve général consistant à trouver des fonctions de mesure sur les configurations telle que cette mesure décroisse le long d'une séquence d'exécution selon un ordre bien-fondé. Des travaux plus récents essaient d'automatiser ce principe et proposent des heuristiques pour la génération des fonctions de mesure dans quelques cas restreints [DGG00, CS01]. Ces travaux raisonnent sur la structure du modèle original. Nos techniques, quant à elles, travaillent sur la structure du graphe symbolique obtenu par l'analyse d'atteignabilité. Cette différence est de taille. En effet, dans notre graphe symbolique, notre technique d'accélération implique que les circuits contenus dans le modèle d'origine soient dépliés jusqu'à ce que nous trouvions une séquence de transitions ayant un effet périodique. Ce déroulement des séquences d'exécution aide non seulement le calcul de l'itération des circuits ($post^*$) mais va aussi nous permettre de raisonner sur l'itérabilité de ces itérations. Ainsi, nos techniques sont

assez puissantes pour traiter des cas complexes pour lesquels si nous raisonnions directement sur le modèle concret, il serait nécessaire de définir une fonction de mesure par rapport à un ordre lexicographique sur les variables, ordre qu'il n'est pas aisé de deviner automatiquement.

Chapitre 6

Analyse des systèmes à file d'attente avec perte

Nous donnons dans ce chapitre la représentation que nous avons choisie pour représenter les contenus de files d'attente avec perte. Cette représentation est un sous-ensemble des expressions régulières.

Nous définissons tout d'abord les expressions régulières simples (SRE : Simple Regular Expression) puis nous discutons de la réalisation des opérations telles que inclusion, intersection sur ces SRE. Puis nous montrons qu'il est alors possible de calculer l'effet de toutes transitions sur une SRE et dans certains cas de pouvoir donner l'effet d'un circuit.

6.1 Préliminaires : langage

Définition 31 (Sous-chaîne cyclique $\preceq_c$)

Soit Σ un alphabet. Soient x et y appartenants à Σ^*.
x est une sous-chaîne cyclique de y (noté $x \preceq_c y$) si et seulement si

$$\exists\, x_1,\ x_2 \in \Sigma^* \text{ tels que } x = x_1 \bullet x_2 \text{ et } x_2 \bullet x_1 \preceq y$$

Exemples

- $abcde \preceq_c cfdfeabf$?
  oui, si nous prenons $x_1 = ab$ et $x_2 = cde$ alors $cdeab \preceq cfdfeabf$.
- $abcde \preceq_c dceab$?
  non, car nous ne pouvons pas trouver de recomposition cyclique de $abcde$ qui soit sous-chaîne de $dceab$, en effet nous pouvons toutes les énumérer :

$x_1 = \varepsilon$	$x_2 = abcde$	:	$abcde \not\preceq cfdfeabf$
$x_1 = a$	$x_2 = bcde$	:	$bcdea \not\preceq cfdfeabf$
$x_1 = ab$	$x_2 = cde$	:	$cdeab \not\preceq cfdfeabf$
$x_1 = abc$	$x_2 = de$	:	$deabc \not\preceq cfdfeabf$
$x_1 = abcd$	$x_2 = e$	:	$eabcd \not\preceq cfdfeabf$
$x_1 = abcde$	$x_2 = \varepsilon$	:	$abcde \not\preceq cfdfeabf$

Proposition 12

La relation $\preceq_c$ peut être décidée en un temps quadratique.

Preuve.
Il suffit d'essayer toutes les décompositions de x, x étant fini.
■

Définition 32 (Sous-chaîne plus $\preceq^+$)
Soit Σ un alphabet. Soient x et y appartenants à Σ^*.
x est une sous-chaîne plus de y (noté par $\preceq^+$) si et seulement si

$$\exists\, m \geq 1 \text{ tel que } x^{m+1} \preceq y^m$$

Exemple :
- *ab* $\preceq^*$ *baba* ?
  Oui, car m=2 et *ababab* $\preceq$ *babababa*.

Proposition 13
Si m existe alors m peut être trouvé dans l'intervalle [1 .. $|y|$].
La relation $\preceq^+$ peut être calculée en un temps quadratique.

Définition 33 (Langage fermé par le bas)
Soit Σ un alphabet. Soit $L \subseteq \Sigma^*$ un langage sur Σ.
L est fermé par le bas (downward closed) si pour tout $x \in L$, s'il existe un $y \preceq x$ alors $y \in L$.

6.2 Expressions Régulières Simples : SRE (Simple Regular Expression)

Avant de définir les Expressions Régulières Simples, nous donnons quelques notations au sujet des langages et expressions réguliers.

Soit Σ un alphabet fini, nous définissons des expressions régulières et les langages générés par ces expressions de manière standard :

Notations :
- Pour une expression régulière r , $[\![r]\!]$ représente le langage défini par r.
- Soient r_1 et r_2 des expressions régulières, on note alors par $r_1 \equiv r_2$ ($r_1 \sqsubseteq r_2$) le fait que $[\![r_1]\!] \equiv [\![r_2]\!]$ ($[\![r_1]\!] \subseteq [\![r_2]\!]$).
- Soient r_1 et r_2 des expressions régulières, $r_1 \sqsubset r_2$ si et seulement si $r_1 \sqsubseteq r_2$ et $r_1 \not\equiv r_2$.
- Pour une expression régulière r, on note par $\lambda(r)$ l'ensemble des lettres de Σ apparaissant dans r.

Nous définissons maintenant les Expressions Régulières Simples. Ce sont des compositions de *produit*, le produit étant lui même une concaténation d'*expression atomique*.

Définition 34 (Expressions Régulières Simples — SRE)
Soit Σ un alphabet fini.
- *Une* **expression atomique** *sur Σ est :*

  - *Soit de la forme* $(a + \epsilon)$*, où* $a \in \Sigma$*.*
  - *Soit de la forme* $(a_1 + a_2 + \cdots + a_n)^*$ *où* $a_1, \cdots, a_n \in \Sigma$*.*
- *Un* **produit** *sur* Σ *(pouvant être vide) est la concaténation* $e_1 \bullet e_2 \bullet \cdots \bullet e_m$ *d'expressions atomiques* $e_1, \cdots, e_m$ *sur* Σ*. On dit que le produit contient alors* m *expressions atomiques. On note par* ϵ *le produit vide.*
- *Une* **expression régulière simple** r *sur* Σ *est de la forme* $p_1 + \cdots p_m$*, où* $p_1, \cdots p_m$ *sont des produits sur* Σ*. Nous utilisons le symbole* $\emptyset$ *pour représenter une expression régulière simple vide : nous supposons donc que* $[\![\emptyset]\!]$ *est le langage vide* $\emptyset$*.*

Un langage L est dit *régulier simple* s'il est représentable par une expression régulière simple.

Nous montrons maintenant que les opérations dont nous avons besoin pour l'analyse des systèmes sont effectivement faisables pour les SRE.

Tout contenu de file est représentable par une SRE.

L'ensemble vide $\emptyset$ est représenté par la SRE vide "$\emptyset$", l'ensemble de tous les contenus possibles est représenté par la SRE $(a_1 + a_2 + \cdots + a_m)^*$ où $a_1, a_2 \cdots, a_m \in \Sigma$ et tous les messages sont représentés.

Le test du vide est décidable, il suffit de regarder si nous avons la SRE $\emptyset$.

Soient s_1, s_2 deux SRE, l'union de ces deux expressions est l'expression régulière simple $s = s_1 + s_2$.

6.2.1 Inclusion pour les expressions régulières simples

Dans cette section nous montrons comment vérifier l'inclusion entre deux SRE. Nous notons l'inclusion entre 2 SRE par le symbole $\sqsubseteq$.

Commençons par voir que si un produit p est inclus dans une SRE alors il existe un produit p' de cette expression qui couvre le produit p.

Lemme 8
Soit Σ *un alphabet. Soient* p*,* p_1*,* $\cdots, p_n$ $n+1$ *produits sur* Σ*. Si* $p \sqsubseteq p_1 + \cdots + p_n$ *alors* $p \sqsubseteq p_i$ *pour un* $i \in [1 \cdots n]$*.*

Preuve. Pour tout entier naturel k, nous pouvons définir une séquence $x \in [\![p_1]\!]$ et $x \notin [\![p_2]\!]$ pour tout produit p_2 si $p_1 \not\sqsubseteq p_2$ et p_2 contenant au plus k expressions atomiques.

La preuve du lemme est alors immédiate. Soit $p = e_1 \bullet e_2 \bullet \cdots \bullet e_m$. Nous définissons $x = y_1 \bullet y_2 \bullet \cdots \bullet y_m$ tel que pour tout $i \in [1 \cdots m]$ y_i est défini de la façon suivante. Si $e_i = (a + \epsilon)$ alors $y_i = a$. Si $e_i = (a_1 + \cdots a_l)^*$ alors $y_i = (a_1 \bullet \cdots \bullet a_l)^{k+1}$. ■

Nous définissons un ordre partiel $\sqsubseteq$ sur les expressions atomiques de telle sorte que :

$$(a + \epsilon) \sqsubseteq (a_1 + a_2 + \cdots + a_m)^* \quad \text{si } a \in \{a_1, a_2, \cdots, a_m\}$$
$$(a_1 + a_2 + \cdots + a_m)^* \sqsubseteq (b_1 + b_2 + \cdots + b_n)^* \quad \text{si } \{a_1, a_2, \cdots, a_m\} \subseteq \{b_1, b_2, \cdots, b_n\}$$

Il est alors maintenant possible de montrer que la vérification de l'inclusion entre 2 produits peut être faite en un temps linéaire.

Lemme 9
L'inclusion de produits peut être vérifiée en un temps linéaire.

Preuve.
En effet, $\epsilon \sqsubseteq p$, $p \not\sqsubseteq \epsilon$ si $p \neq \epsilon$, et $p_1 = e_{11} \bullet p'_1 \sqsubseteq p_2 = e_{21} \bullet p'_2$ si l'un des cas suivant est vérifié :

- $e_{11} \not\sqsubseteq e_{21}$ et $e_{11} \bullet p'_1 \sqsubseteq p'_2$.
- $e_{11} = e_{12} = (a + \epsilon)$ et $p'_1 \sqsubseteq p'_2$.
- $e_{12} = (a_1 + a_2 + \cdots + a_n)^*$, $e_{11} \sqsubseteq e_{12}$ et $p'_1 \sqsubseteq p'_2$.

■

Lemme 10
L'inclusion d'une SRE peut être vérifiée en un temps quadratique.

La preuve se déduit des lemmes précédents.

6.2.2 Forme normale pour les expressions régulières simples

Nous définissons tout d'abord la notion de *produit canonique*. En effet, le produit $p_1 = (a + \epsilon) \bullet (a + b)^*$ et le produit $p_2 = (a + b)^*$ représentent le même langage.

Définition 35 (Produit normal)
Un produit $e_1 \bullet e_2 \bullet \cdots \bullet e_n$ est dit normal si pour chaque i tel que $1 \leq i < n$ nous avons

$$e_i \bullet e_{i+1} \not\sqsubseteq e_{i+1} \ \wedge \ e_i \bullet e_{i+1} \not\sqsubseteq e_i$$

Nous avons alors le lemme suivant.

Lemme 11
Pour tout produit p, il existe une unique forme normale que nous dénotons par $\tilde{p}$ tel que $\tilde{p} \equiv p$. De plus, cette forme normale peut être calculée en un temps linéaire.

Nous pouvons maintenant, transposer les définitions et lemmes ci-dessus aux SRE.

Définition 36 (Expression Régulière Simple normale)
Une SRE $r = p_1 + p_2 + \cdots + p_n$ est dite normale si chacun des produits p_i est normal pour $i \in [1 \cdots n]$ et $p_i \not\sqsubseteq p_j$ pour $1 \leq i \neq j \leq n$.

De la même manière que précédemment, toute expression régulière simple a une forme normale (modulo la commutativité des produits).

Lemme 12
Pour tout SRE r il existe une unique forme normale que nous dénotons $\tilde{r}$ telle que $\tilde{r} \equiv r$. De plus, $\tilde{r}$ peut être calculé à partir de r en un temps quadratique.

6.3 Application d'une transition – $post_t$

Regardons l'effet des quatre opérations possibles dans une transition de système à file d'attente avec perte (?a, !a, nop, vide ?). A partir d'un langage L et d'une opération $op \in \{!a, ?a, nop, vide?\}$, nous allons définir $L \otimes op$ le plus petit langage fermé par le bas tel que $y \in (L \otimes op)$ si il existe un mot x appartenant à L satisfaisant l'une des 4 conditions suivantes :

1. $op = !a$ et $y = x \bullet a$ ou,
2. $op = ?a$ et $a \bullet y = x$ ou,
3. $op = nop$ et $y = x$,
4. $op = vide?$ et $\epsilon \in x$ et $y = \epsilon$.

Lemme 13

Pour une SRE r et une opération $op \in O_f$, il existe une SRE, dénotée $r \otimes op$, telle que $[\![r \otimes op]\!] = [\![r]\!] \otimes op$. De plus, $r \otimes op$ peut être calculée en un temps linéaire.

Preuve. Soit un produit p et une opération $op \in O_f$. Regardons pour chacune des opérations possibles comment construire $p \otimes op$.

- $p \otimes (!a) = p \bullet (a + \epsilon)$.
- $p \otimes (nop) = p$.
- $p \otimes (vide?) = \epsilon$ quelle que soit la composition de p car ϵ appartient à toutes les expressions atomiques (langages fermés par le bas).
- $\epsilon \otimes (?a) = \emptyset$. et
- si $p = e \bullet p_1$ alors

$$p \otimes (?a) = \begin{cases} p & \text{si } e = (a_1 + \cdots a_n)^* \text{ et } a \in \{a_1, \cdots, a_n\} \\ p_1 & \text{si } e = (a + \epsilon) \\ p_1 \otimes (?a) & \text{sinon} \end{cases}$$

Pour une SRE $p_1 + p_2 + \cdots + p_n$ nous avons

$$(p_1 + p_2 + \cdots + p_n) \otimes op = (p_1 \otimes op) + \cdots (p_n \otimes op)$$

■

Ainsi, nous venons de donner l'algorithme pour effectuer une transition à partir de toute représentation de contenu de file d'attente et pour chaque opération possible. Nous venons de donner la définition pour une file d'attente, la fonction $post_t$ devra prendre en entrée plusieurs files d'attente et une opération $op \in O_f$ pour chacune d'elles.

6.4 Effet d'un circuit – $post^*$

Nous l'avons vu au travers de l'exemple du protocole du bit alterné (page 21), il est possible que l'ensemble des configurations atteignables d'un système à file d'attente soit infini à cause de l'itération infinie d'un circuit. Par exemple, dans le protocole du bit alterné, la boucle de l'émetteur étiquetée em_0 peut être appliquée un nombre illimité de fois puisque nous considérons des files d'attente non bornées. Si nous partons de la configuration qui contient ϵ dans la file M, il est alors possible d'atteindre le contenu de file $\overbrace{(m_0 + \epsilon)(m_0 + \epsilon) + \cdots + (m_0 + \epsilon)}^{\text{n fois}}$ quel que soit $n \geq 0$. Nous voudrions en fait exprimer que la file peut contenir le langage $(m_0)^*$.

Nous considérons ici les circuits dans la partie contrôle du modèle. Si *ops* représente l'ensemble des opérations sur les files correspondant au circuit, nous cherchons à calculer

l'effet sur une SRE de l'application un nombre illimité de fois (si possible) de *ops*. Dans le lemme 14, nous montrons que pour chaque SRE et chaque séquence d'opération *ops* il existe un naturel n tel que appliquer le circuit n fois ou plus sur la SRE peut être caractérisé par une autre SRE. En d'autres mots, l'effet de la boucle se "stabilise" après au plus n itérations du circuit, dans le sens où toutes chaines obtenues par itération supérieure sont contenues dans la SRE calculée. Ainsi, l'effet du calcul de l'itération un nombre non borné de fois d'un circuit est représentable par une union d'expressions régulières simples : une d'entre-elles représente toutes les itérations après n tandis que les autres représentent respectivement l'effet du circuit itéré j fois tel que $1 \leq j < n$.

Dans la preuve du lemme nous utilisons les notions de sous-chaines cycliques, sous-chaines plus. Soit $ops = op_1 op_2 \cdots op_n$ une séquence composée de n opérations, L un langage régulier, nous définissons $L \otimes ops$ par $L \otimes op_1 \otimes op_2 \otimes \cdots \otimes op_n$. Nous notons par ops^m la concaténation de m copies de la séquence d'opération *ops*. Par *ops*! (resp. *ops*?), nous dénotons la sous-séquence de *ops* qui ne contient que des opérations d'envoi (resp. de réception). Soit ops_{vide} un bouléen, il est vrai si la séquence d'opérations *ops* contient une opération de test à vide. Pour un produit p, $|p|$ représente le nombre d'expressions atomiques contenu dans p.

Lemme 14

Pour un produit p et une séquence ops d'opération nous avons le résultat suivant. Il existe un naturel n et un produit p' tels que :

- *soit $p \otimes ops^n = \emptyset$.*
- *soit $p' = \cup_{j \geq n} [\![p \otimes ops^j]\!]$.*

De plus p' peut être calculé en un temps quadratique.

Preuve.

Soit $\lambda(ops!) = \{b_1, \cdots, b_k\}$ l'ensemble des messages envoyés par le circuit. La preuve se décompose en 5 cas. Dans les deux premiers la boucle peut être itérée un nombre infini de fois et le contenu des canaux est illimité. Dans les cas 3 et 4 la boucle peut aussi être itérée un nombre infini de fois mais le contenu de la file est borné. Dans le dernier cas, la boucle se bloque après n itérations.

1. Si $(ops?)^* \subseteq [\![p]\!]$ et $ops_{vide} = faux$.
   Nous avons alors deux cas possibles.
   - Soit *ops*? est vide , alors

   $$n = 0 \ \wedge \ p' = p \bullet (b_1 + \cdots b_k)^*$$

   - Soit *ops*? n'est pas vide : soit e la première expression atomique dans le produit p (à partir de la gauche) telle que $\lambda(ops?) \subseteq e$. $p = p_1 \bullet e \bullet p_2$ nous avons alors

   $$n = |p_1| \ \wedge \ p' = e \bullet p_2 \bullet (b_1 + \cdots b_k)^*$$

   Intuitivement, après avoir consommé les mots de p_1, le circuit peut être exécuté un nombre illimité de fois en ajoutant à droite l'ensemble correspondant aux messages contenus dans *ops*!. A cause de la perte de messages, l'effet global est obtenu en concaténant à droite de $e \bullet p_2$ la fermeture par le bas de $(ops!)^*$ qui est précisément $(b_1 + \cdots b_k)^*$.

2. si ops_{vide} = faux, $(ops?)^* \not\subseteq [\![p]\!]$, $ops? \preceq^+ ops!$ et $p \otimes ops \neq \emptyset$, alors

$$n = |p| \ \wedge \ p' = (b_1 + \cdots + b_k)^*$$

Intuitivement , puisque $(ops?)^* \not\subseteq [\![p]\!]$, Le contenu original de la file va être consommé après au plus n itérations. Nous avons $ops? \preceq^+ ops!$ ce qui implique qu'il existe un m tel que $x^{m+1} \preceq y^m$. Donc nous pouvons ajouter à la file au moins $ops!$ toutes les $m+1$ itérations. En itérant le circuit suffisamment, nous pouvons concaténer autant de $ops!$ que nous voulons à la fin de la file. Aussi, par perte de message, l'effet total sera $(b_1 + \cdots + b_k)^*$.
La condition $p \otimes ops \neq \emptyset$ garantit que la première itération du circuit peut être faite. C'est pour couvrir les cas où par exemple la file est vide et l'opération de réception est à faire en premier (dans ces cas là bien que les premières conditions soient vérifiées nous ne pouvons pas itérer le circuit).

3. si ops_{vide} = faux,$(ops?)^* \not\subseteq [\![p]\!]$, $ops? \not\preceq^+ ops!$, $ops? \preceq_c ops!$ et $p \otimes ops^2 \neq \emptyset$, alors

$$n = |p| \ \wedge \ p' = p \otimes ops^{n+1}$$

Bien que nous puissions itérer autant de fois que nous voulons le circuit, le contenu de la file ne grossira plus après la n-ième itération. Nous demandons que le circuit puisse se faire au moins 2 fois ($p \otimes ops^2 \neq \emptyset$), car la condition du cas 2 n'est plus suffisante. Voici un petit contre-exemple :
soit $p = ba$ et $ops = (?b)(?a)(!a)(!b)$, nous avons $p \otimes ops = ab$ mais $p \otimes ops^2 = \emptyset$.
Cela vient du fait que pour les chaînes x et y, la relation $x \preceq^+ y$ implique que $x \preceq y$, alors que $x \preceq_c y$ implique elle que $x \preceq y^2$ et non pas $x \preceq y$.

4. si ops_{vide} = vrai, $p \otimes ops^2 \neq \emptyset$, alors

$$n = 1 \ \wedge \ p' = p \otimes ops$$

Intuitivement, si la boucle contient un test à vide, le résultat de la boucle sera toujours le résultat des opérations appliquées à partir du vide après la dernière opération vide. Il nous faut vérifier que la boucle peut s'exécuter au minimum deux fois, la première pour vérifier que le contenu de la file sur lequel nous voulons accélérer ne bloque pas tout de suite la boucle puis une deuxième fois pour vérifier que le résultat de la boucle ne se bloque pas lui-même.

5. si les conditions 1, 2, 3 et 4 ne sont pas satisfaites alors

$$n = |p| + 1 \ \wedge \ p \otimes ops^n = \emptyset$$

Dans ce cas, le circuit peut être exécuté au plus n fois, après la file est vide, nous arrivons dans un état de bloquage à cause de l'impossibilité d'exécuter des opérations de réception.

■

Dans le cas où nous avons une SRE (à la place d'un produit) il est alors possible d'appliquer le lemme précédent à chacun des produits séparément.

Dans le cadre de plusieurs files d'attente, l'algorithme applique le lemme précedent à chacune des files séparément, la boucle est infiniment exécutable si et seulement si la boucle peut être exécutée infiniment souvent pour chacune des files d'attente respectivement. Si le circuit peut être exécuté un nombre illimité de fois, le résultat final pour le contenu des files est le produit cartésien des expressions calculées par le lemme 14.

Remarque :

Les circuits traités par cet algorithme sont des circuits simples. En fait, la plupart des méthodes utilisées (cf. section 6.6) pour les files d'attente ne traitent que les circuits simples ou des systèmes particuliers : les systèmes de communication en anneau. Un système en anneau met en œuvre une seule file d'attente et chaque processus composant le système ne communique qu'avec ses voisins proches (pour cela il faut imaginer l'ensemble des processus disposés en cercle et ne communiquant qu'avec leurs voisins de droite et de gauche).

Voici un exemple (présenté par Bouajjani dans [Bou01]) que nous ne pouvons traiter et qu'à notre connaissance personne ne peut traiter. Il s'agit en effet de trouver un ensemble infini en ne produisant que des ensembles finis.

Le système que nous considérons est composé d'un processus. Ce processus contient un état de contrôle, une seule file d'attente et 2 transitions représentées par la figure 6.1. Le vocabulaire des mots de la file est $\Sigma = \{a, b\}$. Le contenu de la file est initialisé à $(a + \epsilon)$.

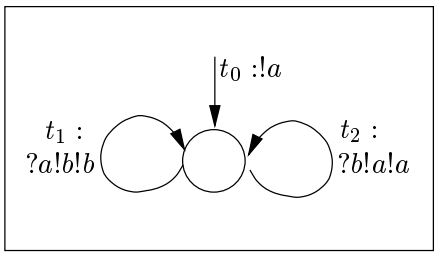


FIG. 6.1 – Un exemple qui ne termine pas : le modèle

La figure 6.2 représente une partie de l'ensemble des configurations atteignables. Nous notons par ab, l'expression régulière $(a+\epsilon)\bullet(b+\epsilon)$ et par b^3 l'expression $(b+\epsilon)\bullet(b+\epsilon)\bullet(b+\epsilon)$.

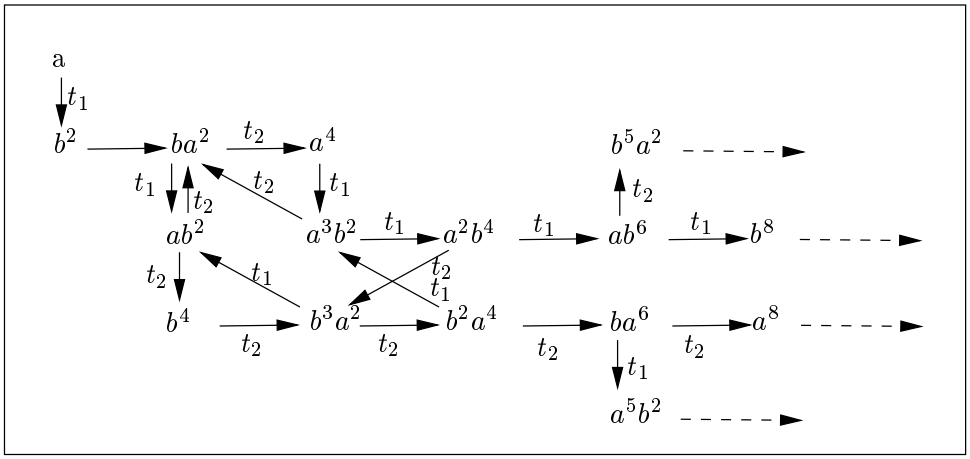


FIG. 6.2 – Un exemple qui ne termine pas : une partie des configurations

Examinons maintenant plus en détail la raison de l'infinitude de cet exemple. Le langage défini par le modèle est $L = a^*b^* + b^*a^*$. Nous montrons que quel que soit le circuit considéré et quel que soit le mot $w \in \Sigma^*$ que nous prenions l'itération de ce circuit est bornée et nous obtenons un ensemble fini de mots.

Dans notre modèle les circuits potentiels sont de la forme suivante :

$$\theta = t_1^{n_1} t_2^{m_1} t_1^{n_2} t_2^{m_2} \cdots t_1^{n_k} t_2^{m_k}$$

où $k > 0$ et $n_i, m_i > 0$ pour tout $i \in [1, k]$.

Regardons tout d'abord l'effet du circuit $\theta = t_1$ puis $\theta = t_2$ sur le mot $babab$, nous obtenons alors

$$t_1^*(babab) = bab^3 + bb^4$$
$$t_2^*(babab) = ababa^2 + aba^4 + a^6$$

En fait, le nombre d'itérations possibles pour la transition t_1 (resp. t_2) est toujours borné. Il dépend du nombre d'occurences de la lettre a (resp. b) dans le mot initial.

Maintenant, si nous prenons $\theta = t_1 t_2$ et le mot initial $w = a$ nous voyons sur la figure 6.2 que $\theta^*(w) = ba^2$.

Ainsi, le langage défini par le modèle contient un nombre infini de mots, or nous ne générons que des ensembles finis de mots, il est donc impossible d'obtenir la terminaison de ce système.

6.5 Construction du graphe symbolique

Ainsi, nous venons de définir une représentation pour les contenus de files et les algortihmes $post_t$ et $post^*$ permettant de calculer respectivement l'effet d'une transition et d'un circuit.

Nous pouvons les utiliser et appliquer l'algorithme 3.4 dans l'exemple du protocole du bit alterné.

La configuration initiale est $\gamma_0 = ((e_0, r_0), \epsilon, \epsilon)$. L'ensemble des configurations atteignables est représenté par le tableau ci-dessous, le graphe symbolique correspondant est quant à lui représenté par la figure 6.3.

état émetteur	état récepteur	file M	file A
e_0	r_0	m_1^*	a_1^*
e_1	r_0	$m_1^* m_0^*$	a_1^*
e_1	r_1	m_0^*	a_1^*
e_1	r_2	m_0^*	$a_1^* a_0^*$
e_2	r_2	m_0^*	a_0^*
e_3	r_0	m_1^*	$a_0^* a_1^*$
e_3	r_2	$m_0^* m_1^*$	a_0^*
e_3	r_3	m_1^*	a_0^*

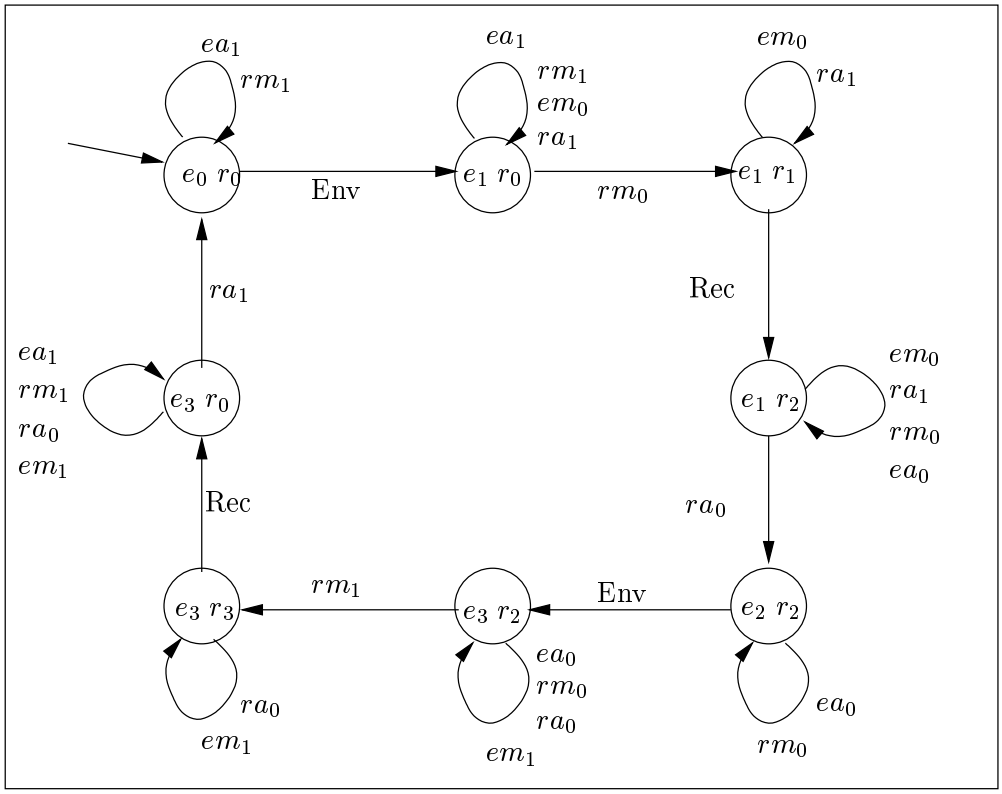


FIG. 6.3 – Graphe symbolique pour le protocole du bit alterné

6.6 Discussion

Il existe plusieurs autres représentations pour le contenu des files de systèmes à files parfaites. Dans ce cadre, le problème de l'atteignabilité est indécidable.

– Pachl [Pac87] est le premier à notre connaissance à proposer de représenter l'ensemble des configurations atteignables par un ensemble reconnaissable. Un ensemble reconnaissable étant une union finie de produits cartésiens d'ensembles réguliers. Pachl n'a pas donné d'algorithmes efficaces pour la manipulation de cette représentation.

- Dans [FM96], Finkel et Marcé proposent une procédure d'analyse symbolique utilisant une classe d'expressions régulières qui ne sont pas comparables aux SRE. De plus, l'ensemble des configurations atteignables calculé par cette procédure n'est pas toujours exact.
- Boigelot et Godefroid [BG96, BGWW97] utilisent des automates finis (nommés QDD : Queue Decision Diagram) qui représentent le contenu des files par des ensembles reconnaissables. Dans [BGWW97] il a été montré que l'effet de toute boucle est représentable et calculable pour un système ne contenant qu'une seule file d'attente. Si l'on a deux files d'attente, l'effet du circuit n'est plus représentable par les QDD. Ceci est dû au fait que l'exécution répétée d'une boucle peut créer des liens entre le nombre de symboles contenus dans les différentes files d'attente. Dans [BGWW97] il est donné une complète caractérisation des sortes de circuits qui préservent la reconnaissance.
- Pour représenter les liens entre les contenus des files, une structure de représentation telle que les CQDD ("constrained QDD"), combinant les automates finis avec des contraintes arithmétiques linéaires est nécessaire. Cette structure a été présentée par Bouajjani et Habermehl dans [BH97, Hab98].
- Sutre et Finkel ont utilisé quant à eux des expressions régulières semilinéaires (appelées SLRE) [Sut00, FIS00]. Les SLRE sont faciles à manipuler, elles sont à l'intersection des langages représentables par les QDD et par les CQDD. Elles ne permettent donc pas d'analyser des systèmes qui comptent car les liens sur les contenus de files créés par les itérations ne sont pas pris en compte.
- Dans le cadre des systèmes parfaits, la restriction à l'analyse de systèmes dits "token ring" permet une analyse finie [Céc99, ABB01]. L'ensemble des configurations atteignables est alors effectivement constructible.

On peut aussi analyser des systèmes à files d'attente avec duplication de messages ou insertion de messages [CFP96, Céc98].

Nous avons choisi pour notre part d'analyser des systèmes ayant des files d'attente avec perte de messages, ainsi nous pouvons ne pas considérer les liens entre le nombre de messages dans les différentes files grâce à la perte de messages ce qui simplifie le calcul des effets des boucles. Utiliser les SRE dans le cadre des systèmes à files d'attente avec perte offre encore d'autres avantages.

Tout d'abord les opérations sur les QDD et les CQDD sont de complexité exponentielle et même encore $NP \cap coNP$ pour les SLRE. Nous avons pour les SRE des algorithmes de complexité polynomiale.

Ensuite, il existe une forme normale pour les SRE et calculer cette forme normale pour une expression donnée est exécutable en un temps polynomial. Alors que les QDD admettent une forme normale via minimisation des automates (mais demandent un temps plus élevé), les CQDD n'ayant pas de forme canonique.

Enfin, représenter dans la sémantique du modèle la perte des messages évitent d'alourdir le modèle concret avec des boucles emboîtées simulant cette perte dans le cas des files parfaites. Car ajouter des boucles emboîtées amène généralement à ne plus pouvoir analyser le système.

Chapitre 7

TReX : un outil d'analyse de systèmes infinis

Les méthodes et algorithmes que nous avons détaillés dans les chapitres précédents sont implantés dans un prototype appelé TReX pour "*Tool for Reachability Analysis of Complex Systems*". Cet outil permet l'analyse automatique de systèmes manipulant des variables de types différents et des paramètres (variables non instanciées). Nous avons pour le moment, implanté les systèmes à compteurs paramétrés, les systèmes à horloges paramétrées, les systèmes à files d'attente et toutes les combinaisons de systèmes provenant de ces trois domaines de définition.

Nous commençons ce chapitre en explicitant l'environnement de TReX. En effet, nous avons conçu TReX avec un souci d'interconnectivité avec les outils pré-existants dans le domaine de la vérification [AAB+99a]. Le langage d'entrée choisi est le langage IF [BFG+99a, BFG+99b] servant de passerelle entre les langages de description formelle tels que SDL ou Lotos et les outils de vérifications. Les sorties de TReX , à savoir notamment le graphe symbolique et l'ensemble des configurations atteignables, peuvent être utilisés comme entrées d'outils comme CADP [FGK+96] ou InVeST [BLO98]. Grâce à la boite à outils CADP nous pouvons vérifier des propriétés de sûreté sur le graphe symbolique fini donné par TReX. L'ensemble des configurations atteignables peut servir d'abstraction pour une analyse réalisée avec InVeST.

Nous entrons ensuite dans les détails de TReX . Nous voyons que nous avons construit TReX de façon générique. Finalement, nous donnons quelques performances sur quelques exemples puis nous comparons notre outil à quelques outils de vérification existants.

7.1 Architecture de TReX

Voici une vision d'ensemble de TReX. Nous entrons dans les détails dans les sections suivantes.

7.1.1 Vue générale

La figure 7.1 représente l'environnement et l'architecture de TReX.

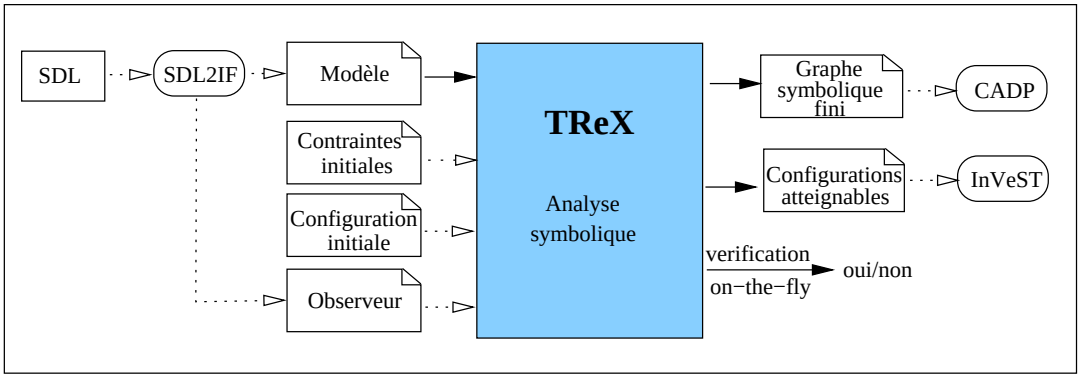


FIG. 7.1 – Vue générale de l'environnement et l'architecture de TREX

TREX est un outil en connexion avec les outils et les formalismes de description des systèmes de la manière suivante. Le format de description des systèmes que nous avons adopté est le format IF (cf. section 7.2.1). Cela nous permet de pouvoir analyser des systèmes dont la description est écrite en SDL (langage de description des systèmes, notamment protocoles de communication, utilisé dans les entreprises). Les sorties de notre outil sont, d'un côté, l'ensemble des configurations atteignables du système qui peut alors être utilisé comme abstraction pour des outils tels que InVeST. D'autre part, nous obtenons aussi le graphe symbolique fini représentant la sémantique du système de départ sur lequel nous pouvons appliquer des techniques de model-checking fini et des outils tels que CADP ou SPIN pour vérifier notamment des propriétés de sûreté sur les graphes finis.

Ainsi, le modèle d'entrée de TREX est un modèle décrit en IF sur lequel nous appliquons les techniques décrites dans les chapitres précédents. Nous pouvons alors préciser la configuration de départ et les contraintes initiales pour les paramètres éventuels apparaîssant dans le système à analyser. Nous pouvons aussi analyser une propriété de sûreté à la volée en donnant à l'outil la représentation de la propriété à vérifier sous forme de système de transition étiqueté exprimé sous le format IF. Pour plus de précision sur les options de l'outil, on peut se référrer à l'annexe C.1 (page 167).

Nous ne traitons pas pour le moment la vérification de propriété de vivacité, c'est en cours d'implantation.

7.1.2 Vue intérieure

La figure 7.2 représente les éléments composant notre outil. La caractéristique principale de TREX est sa généricité. En effet, trois grands modules composent TREX.

Chaque structure de données est un module à part entière sur lequel est implanté notamment l'algorithme $Post$, $Post^*$, les opérations d'inclusion et d'intersection, le test à vide.

Pour certaines de ces opérations, nous avons besoin de procédure de décision externe à notre outil. Ceci concerne la manipulation des formules arithmétiques, nous revenons sur ce point dans le paragraphe 7.2.2.1.

Enfin, nous avons un module exprimant le calcul des états atteignables générique à toutes les structures de données. L'algorithme en profondeur implanté utilise les fonctions telles que $Post$, $Post^*$ prédéfinies pour chacune des structures de données.

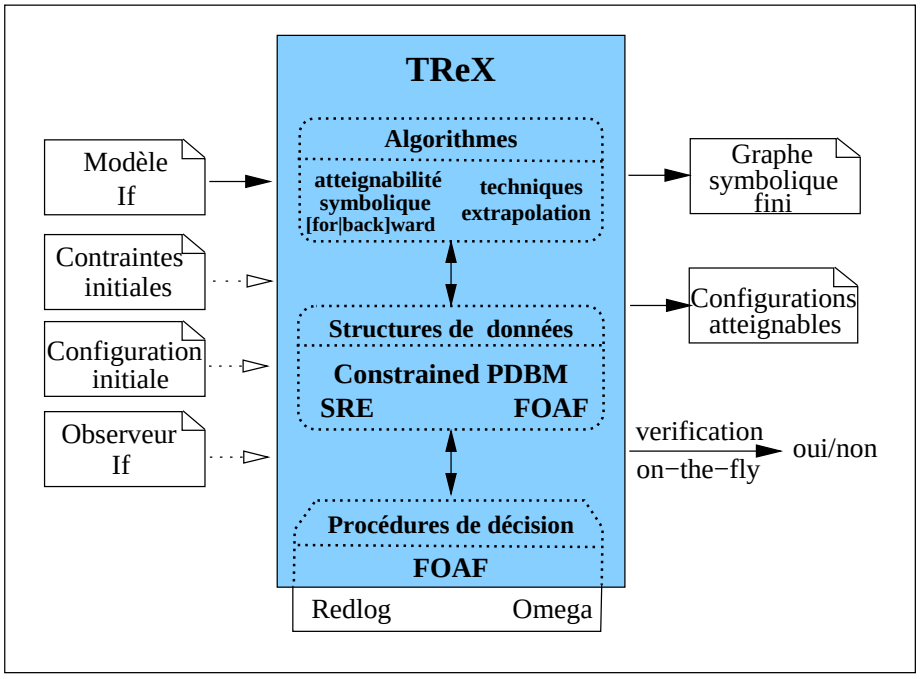


FIG. 7.2 – Vue plus détaillée de TREX

7.2 Description des composantes

TREX est un prototype implanté en C++ contenant environ 56300 lignes de code. Nous détaillons dans cette partie quelques éléments de l'implantation.

7.2.1 Le format d'entrée

Les modèles que nous considérons sont des automates étendus complexes ayant la possibilité de manipuler des compteurs, des horloges, des files d'attente avec perte et des variables booléennes et énumératives. Comme nous l'avons dit précédemment nous utilisons le format IF comme format d'entrée pour nos modèles. Il s'agit en fait d'un sous-ensemble de IF dont nous donnons la grammaire en annexe (annexe C.2, page 169). La version de IF que nous utilisons ne contient pas de labels pour les transitions, ni de notion de paramètres. Pour contourner ces problèmes, nous avons adopté les conventions suivantes :

- Les étiquettes des transitions que nous considérons sont tout message envoyé sur une file d'environnement du système. C'est une file de type *toenv* à déclarer dans le système.
- Nous considérons comme paramètres les variables déclarées dans le système qui ne sont jamais (dans aucune transition) affectées. Un paramètre pour les horloges devra être déclaré de type réel et un paramètre pour les compteurs de type entier.

Ainsi, avant de commencer à proprement parler de notre analyse, nous devons analyser l'arbre abstrait généré par le compilateur IF sur le modèle initial pour en extraire le type du système (c'est-à-dire connaître le type des données manipulées par le système), les

étiquettes des transitions et les différents paramètres.

7.2.2 Matrices de bornes paramétrées contraintes

Une part importante de l'implantation a été la mise en œuvre des matrices de bornes paramétrées contraintes. En effet, ces matrices peuvent très vite devenir lourdes à manipuler car :

- Chaque élément des matrices est un terme arithmétique dont la représentation est cruciale. Nous devons notamment pouvoir les comparer et les simplifier rapidement et simplement. De plus, les formules arithmétiques dont nous cherchons la satisfaction pour les opérations de canonisation ou intersection de matrices utilisent ces éléments. En effet, ces contraintes sont des formules booléennes impliquant des comparaisons de la forme $t \prec t'$ où t et t' sont des expressions arithmétiques (linéaires ou non-linéaires) sur l'ensemble des paramètres et des variables d'itération, et $\prec \in \{<, \leq\}$. Nous devons ici aussi être capables de représenter ces formules de manière compacte et de pouvoir simplifier les termes des formules notamment pour essayer de rester dans le cadre de la non-linéarité. Nous montrons que la simplification des termes peut en effet réduire les cas de non-linéarité nous permettant ainsi d'utiliser des procédures de décision pour le cas linéaire, ce qui est plus simple dans le cadre des réels et est la seule possibilité dans les cadres des entiers.
- L'algorithme de canonisation utilisé sur les matrices est une extension de l'algorithme de calcul du plus court chemin de Floyd-Warshall. Malgré un coût dans le pire des cas très élévé, nous montrons qu'en fait la réelle complexité est moindre grâce aux contraintes sur les paramètres. Pourtant le coût de la canonisation reste consommateur de temps et de mémoire surtout parce que cette opération est utilisée après chaque opération faisant intervenir des matrices. Pour limiter le temps consommé par cette opération nous avons introduit une notion de hachage sur les matrices canonisées : nous conservons les calculs exécutés pour pouvoir les réutiliser.

Nous reprenons chacun des points cités ci-dessus en explicitant les méthodes que nous avons utilisées.

7.2.2.1 Représentation des termes arithmétiques

Tout d'abord quelle que soit la représentation des termes que nous choisissons, nous devons avoir recours à des procédures de décision annexes à notre outil pour décider de la validité ou de la satisfaisabilité de formules arithmétiques. Dans ce but, nous avons choisi d'utiliser les procédures de décision offertes par OMEGA [Tea96] pour les formules arithmétiques sur les entiers et par REDUCE [Hea99] pour les formules sur les réels. L'utilisation des ces procédures de décision est transparente pour l'utilisateur de TREX la seule contrainte imposée est d'avoir installé les outils OMEGA et REDUCE.

Dans un premier temps, nous avions représenté les termes arithmétiques par des chaînes de caractères. Il était alors impossible de les simplifier ou de les comparer autrement qu'en appelant une fonction de simplification externe à TREX. La mémoire utilisée et le temps d'exécution de TREX étaient alors élévés.

Il nous fallait donc trouver une meilleure représentation des formules qui puisse nous permettre de réduire le nombre d'appel aux procédures extérieures. C'est pour cela que nous avons implanté dans TReX une librairie de manipulation de formules arithmétiques (librairie FOAF implantée par Mihaela Sighireanu). De plus, nous avons aussi implanté dans notre libraire des procédures d'élimination de quantificateurs (algorithmes de Fourier-Motzkin) pour les réels nous permettant de réduire encore le nombre d'appel à REDUCE.

Un autre problème intéressant concernant les formules arithmétiques est dû à l'élimination de quantificateurs sur les réels. TReX utilise, nous l'avons dit, deux procédures pour les réels : celle fournie par REDUCE manipulant des termes non-linéaires et une autre fournie par notre librairie FOAF ne manipulant que des termes linéaires sur les variables à éliminer. Un point important dans le cadre de ces procédures est que la complexité de leurs résultats dépend fortement de l'ordre dans lequel les variables sont éliminées. Considérons par exemple la formule ϕ (apparaissant dans l'analyse du BRP) suivante :

$$\exists(T_1, T_2, T_d, synch).\ 2 * max * T_1 - T_2 + 3 * T_d \leq 0 \wedge$$
$$synch - T_2 > 0 \ \wedge\ T_1 - T_d > 0 \ \wedge\ T_1 \geq 0 \ \wedge\ T_2 > 0 \ \wedge\ T_d > 0$$

où max est une variable entière et $T_1, T_2, T_d, synch$ sont des variables réelles. Si l'ordre des variables pour l'élimination des quantificateurs est $synch < T_d < T_2 < T_1$ (c'est-à-dire $synch$ est éliminée la première, puis T_d, etc.), la formule obtenue par REDUCE est

$$max - 2 \geq 0$$

Cette formule est linéaire sur les entiers donc décidable.

Par contre si nous utilisons l'ordre inverse ($synch > T_d > T_2 > T_1$), nous obtenons avec REDUCE la formule suivante :

$$2 * max^2 + 3 * max > 0 \ \wedge\ max - 2 \geq 0$$

Cette formule est équivalente à la première mais elle est non-linéaire et donc non décidable sur les entiers.

Aussi pour essayer de limiter le nombre de cas non-linéaires nous avons implanté dans TReX une heuristique qui décide d'un ordre sur les variables tel que la génération de formules non-linéaires par la procédure de Fourier-Motzkin soit minimisée. Cette heuristique donne aussi de bons résultats pour l'algorithme utilisé par REDUCE.

Nos expériences ont montré que la procédure de Fourier-Motzkin produit moins de formules non-linéaires que l'algorithme de REDUCE. Par exemple, notre procédure appliquée à la formule ϕ avec le second ordre sur les variables donne la formule

$$2 * max + 3 > 0 \ \wedge\ max - 2 \leq 0$$

Cette formule est équivalente à celles données par REDUCE mais est linéaire.

En résumé, notre procédure permet de générer moins de non-linéarité que celle de REDUCE mais l'algorithme de REDUCE est plus général (puisqu'il peut être utilisé pour des formules non-linéaires) et donne de meilleures performances de temps que notre implantation (cf. section 7.3).

7.2.2.2 Algorithme de canonisation

L'algorithme de canonisation que nous avons implanté est une extension de l'algorithme de calcul de la fermeture d'une matrice de Floyd-Warshall pour les matrices de bornes paramétrées contraintes. Dans cet algorithme nous devons pouvoir décider du minimum de deux bornes paramétrées contraintes. Nous reprenons les cas définis lors de la définition du minimum entre deux bornes paramétrées contraintes (cf. définition 27, page 54).

Nous vérifions à chaque pas que la contrainte sur les paramètres reste valide. Pour cela nous utilisons la fonction $ajouter\_contrainte((M,\phi),\phi')$ qui permet de vérifier la satisfaisabilité de la formule $\exists p_i.\exists n_i.\ \phi(p_i,n_i) \ \wedge \ \phi'(p_i,n_i)$. Si elle est satisfaisable la contrainte associée à la matrice M est alors $\phi \wedge \phi'$. De plus nous vérifions à chaque pas que les termes des éléments M_{ii} de la matrice ne sont pas négatifs. En effet, un de ces termes négatifs voudrait dire que le poids sur le chemin pour aller de la variable x_i à la variable x_i est négatif. Nous pouvons dans ce cas prendre infiniment ce chemin pour avoir des poids de plus en plus petits, en fait cela prouve que l'ensemble des valuations exprimé par la matrice est inconsistant.

Nous donnons l'algorithme complet en annexe (page 171).

Remarque : Si la matrice initiale ne contient pas de paramètres, nous revenons à l'algorithme initial. En effet, avec des valeurs pour les termes t_1, t_2, il n'y a que trois cas possible $t_1 < t_2$ ou $t_1 = t_2$ ou $t_1 > t_2$ et non pas un cumul de cas.

Si nous calculons le coût de cet algorithme en terme de nombre de matrices obtenues comme résultat de la canonisation, nous voyons que dans le pire des cas ce coût est de $3^{(taille^3)}$ (les trois boucles imbriquées impliquent $taille^3$ appels à la fonction min qui elle même, dans le pire des cas, peut donner 3 matrices). Pourtant, nous n'obtenons jamais le pire des cas grâce aux contraintes sur les paramètres. Par exemple, pour 2 compteurs (matrice carrée de dimension 3) le pire des cas est de 3^{27}, or en comptant le nombre de matrices obtenues après la canonisation de la matrice représentée ci-dessous (où a, b, c, d, e, f sont des paramètres sur lesquels nous ne donnons pas de contrainte initiale) nous obtenons en fait 127 matrices résultats.

$$\begin{pmatrix} (\leq,0) & (\leq,a) & (\leq,b) \\ (\leq,c) & (\leq,0) & (\leq,d) \\ (\leq,e) & (\leq,f) & (\leq,0) \end{pmatrix}$$

7.2.2.3 Hachage pour la canonisation

Pourtant la canonisation est une opération que nous utilisons après chaque manipulation de matrices contraintes et la canonisation prend du temps. Après avoir remarqué que lors d'une analyse nous effectuions plusieurs fois la même canonisation nous avons voulu conserver les calculs effectués pour ne pas avoir à refaire ces calculs. Pour cela remarquons tout d'abord qu'au cours de l'algorithme de canonisation chaque contrainte sur les paramètres peut éventuellement être restreinte mais pas ouverte, c'est-à-dire qu'il est

possible d'ajouter des contraintes aux formules et non pas d'en enlever.

Propriété 7

Soit la matrice de bornes paramétrées contrainte $\tilde{M} = (M, vrai)$ *telle que l'ensemble des matrices résultant de sa canonisation est le suivant :* $\{\tilde{M_1} = (M_1, \phi_1), \tilde{M_2}, \cdots, \tilde{M_n}\}$. *Alors soit* $\tilde{M'} = (M, \phi)$ *une matrice contrainte ayant la même matrice* M *que* $\tilde{M}$, *le résultat de la canonisation pour* $\tilde{M'}$ *est l'ensemble :*

$$\{(M_1, \phi_1 \ \wedge \ \phi), (M_2 \ \wedge \ \phi), \cdots, (M_n, \phi_n \ \wedge \ \phi)\}$$

Nous avons donc implanté une table de hachage sur les matrices telle que pour toute demande de canonisation d'une matrice (M, ϕ) nous cherchons tout d'abord à savoir si la matrice $(M, vrai)$ a déjà été canonisée. Si oui, nous récupérons le résultat dans la table de hachage, les résultats voulus étant un parcours de liste avec des conjonctions de formules. Si non, nous effectuons la canonisation de $(M, vrai)$ pour la mettre dans notre table de hachage.

En fait, nous ne canonisons pas $(M, vrai)$ mais (M, ϕ_{init}) où ϕ_{init} est la contrainte initiale sur les paramètres qui a été fournie au début de l'analyse. Nous sommes sûrs que les paramètres doivent satisfaire au moins cette contrainte quelle que soit la configuration que nous obtenons lors de l'analyse. Ce dernier point nous permet de réduire le nombre de matrices résultant de la canonisation.

Nous pouvons maintenant ajouter à l'algorithme le traitement de la table de hachage (cf. figure 7.3). Nous notons dans cet algorithme la table de hachage par Th.

```
fonction Canonisation(Matrice contrainte (M_0, φ_0)) : liste de mat. contraintes
  lexique :
    i, j, k, nbcasesocc : entier (0, 0, 0, 0, 1)
    T : tableau de matrices contraintes
    l, li, lj : entier
    lm, lf : liste de matrices contraintes (liste vide, liste vide)
  algo. :
    si (lm := Th[M]) = ∅
    alors /* la matrice n'a pas été canonisée */
      T[0] := (M_0, φ_init)
      tant que ( (k < taille) et puis (nbcasesocc ≠ 0) )
        i := 0
          [...]
        k++
      ftq
      Th(M) := liste composée des éléments de T
      pour toutes les cases l de T occupées :
        lf := ajout_queue(lf, (T[l].M, T[l].φ ∧ φ_0))
    sinon /* la matrice a été canonisée */
      pour toutes les matrices (M, φ) de lm :
        lf := ajout_queue(lf, (M, φ ∧ φ_0))
    retourner lf
```

FIG. 7.3 – Algorithme de canonisation de matrices de bornes paramétrées contraintes avec hachage

7.3 Résultats et performances

Pour montrer l'évolution de TREX et les avantages de chacun des choix d'implantation que nous avons fait, voici quelques petits exemples d'école. Nous montrons ici un exemple de système à compteurs (l'ascenseur, l'algorithme du ticket), de système à horloges (algorithme d'exclusion mutuelle de "Fischer") ou de files d'attente (protocole du bit alterné). Pour chacun de ces exemples nous donnons tout d'abord la description informelle de leur comportement. Si c'est un exemple déjà introduit dans un des chapitres précédents nous ne redonnons ici que le dessin du modèle. Nous donnons ensuite les performances en temps et en mémoire utilisée pour quelques versions de notre outil :

- TREX_1 : représentation des termes par FoafFormula.
- TREX_2 : hachage de la canonisation.
- TREX_3 : élimination des quantificateurs par notre librairie.

On note par $\perp$ lorsque l'exécution a été stoppée par manque de mémoire ou manque de temps. On note par - lorsque l'exécution de l'exemple n'apporte rien. Sauf indications contraires, les exécutions ont été faites sur une machine UltraSparc sun5 avec 1024Mb de mémoire. Les temps sont indiqués en secondes et la mémoire en Mb.

7.3.1 Un ascenseur

Cet exemple peut être trouvé dans [Val89]. Cet exemple représente le fonctionnement d'un ascenseur au travers du fonctionnement de son moteur et du contrôle des étages. Ces deux éléments sont modélisés par des systèmes à compteurs. Ils communiquent au travers de deux compteurs (variables globales). La figure 7.4 représente le comportement de l'ascenseur.

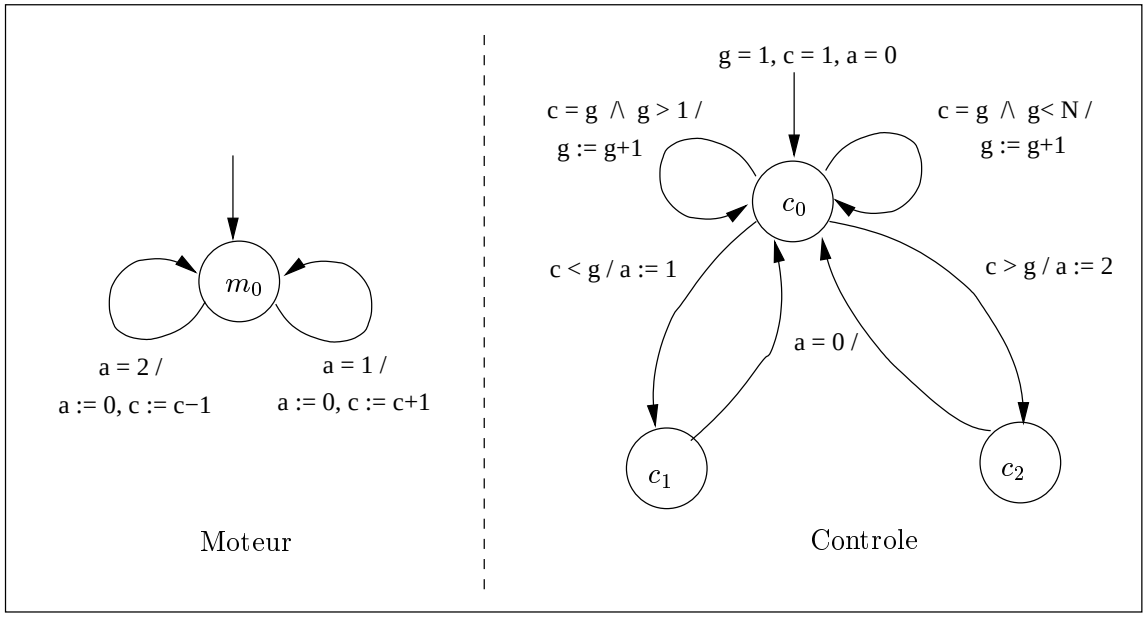


FIG. 7.4 – Modèle de l'ascenseur

La variable globale c contient à tout moment le numéro de l'étage auquel se trouve l'ascenseur. La variable g représente le numéro de l'étage où doit se rendre l'ascenseur.

Le comportement du moteur est le suivant. A tout moment le moteur attend des ordres provenant de la partie contrôle. Ces ordres peuvent être :

- *Monte d'un étage* : la variable c doit alors être incrémentée de 1.
- *Descend d'un étage* : la variable c doit alors être décrémentée de 1.

Ces ordres sont représentés par la variable globale a qui doit être mise à jour correctement par la partie contrôle en fonction de l'étage recherché.

Le comportement de la partie contrôle est le suivant. Elle compare la valeur de c (étage courant) avec celle de g (étage recherché) et envoie les bons ordres au moteur jusqu'à ce que ces deux valeurs soient égales. Ensuite elle choisit un nouveau étage arbitrairement.

Le nombre total d'étages est donné par un paramètre N.

Cet exemple est intéressant car nous pouvons l'analyser en instanciant la valeur de N ou en gardant le paramètre. Le tableau 7.1 résume les résultats que nous avons obtenus dans différents cas :

TAB. 7.1 – Statistique des performances pour l'ascenseur (1)

N	Sans accél.			TREX$_1$			
	Mém.	Tps	nb. conf.	Mém.	Tps e	Tps s	Tps t
2	8.1	0.11	8	5	0	0.12	0.12
3	8.1	0.20	15	6.8	48.81	13.93	62.74
4	8.1	0.24	22	7	52.35	14.99	67.34
5	8.1	0.43	29	7	52.04	15.10	67.14
6	8	0.56	36	7	52.60	14.61	67.21
7	8	0.71	43	7	52.58	15.57	68.15
8	8	1.92	50	7	52.97	14.94	67.91
9	8	1.08	57	7	52.38	15.13	67.51
10	8	1.25	64	7	51.96	15.33	67.29
100	11.3	83.62	694	7	52.96	15.62	68.58
1000	$\perp$		$>$	7	53.15	15.67	68.72
10000	$\perp$			7	53	15.29	68.29
N	-	-	-	7.2	61.44	17.05	78.49
∞	-	-	-	6	14.91	5.43	20.34

TAB. 7.2 – Statistique des performances pour l'ascenseur (2)

N	TREX$_2$				nb. conf.
	Mém.	Tps e	Tps s	Tps t	
2	8.2	0	0.14	0.14	8
3	8.2	6.51	2.44	8.95	8
4	8.2	9.99	4.19	14.18	8
5	8.2	9.06	2.95	12.01	8
6	8.2	8.92	2.94	11.86	8
7	8.2	9.05	2.96	12.01	8
8	8.2	8.74	3.11	11.85	8
9	8.2	8.78	3.01	11.79	8
10	8.2	8.73	3.14	11.87	8
100	8.2	8.92	3.07	11.99	8
1000	8.2	9.06	3.10	12.16	8
10000	8.2	8.84	3.05	11.89	8
N	8.3	9.46	3.65	13.11	9
∞	8.1	4.37	2.04	6.41	7

Nous remarquons que l'analyse de cet exemple est constante en temps et en mémoire quel que soit le nombre d'étages considéré. Aussi, si nous laissons libre le paramètre N (avant dernière ligne du tableau) nous calculons en une fois l'analyse de l'ascenseur pour tout nombre d'étage donné. Ce calcul est fait en 13s pour 9 configurations accessibles alors que, par exemple, pour traiter le cas de 10 étages (sans accélération) il nous avait fallu 83s et 694 configurations.

7.3.2 Exclusion mutuelle : algorithme du "Bakery"

Cet exemple d'algorithme d'exclusion mutuelle manipulant des compteurs peut être trouvé dans [And91, BGP99].

Nous avons introduit cet exemple d'algorithme d'exclusion mutuelle dans le paragraphe 4.5.1 (page 74). Pour un système composé de n processus, il contient alors n compteurs que nous nommons ai ($i \in [0, n]$). La figure 7.5 reprend la modélisation d'un processus.

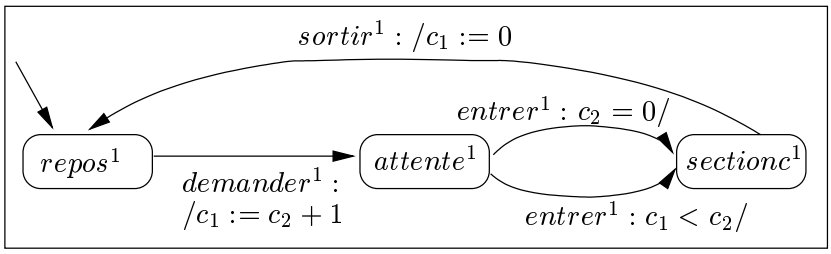


FIG. 7.5 – Modélisation d'un processus pour le "Bakery"

Les résultats de l'analyse de cet exemple pour deux processus sont représentés dans le tableau 7.3, nous avons obtenu 33 configurations. Une erreur dans Omega ne nous permet pas de donner de résultats pour des exemples contenant plus de deux processus.

TAB. 7.3 – Statistique des performances pour l'analyse du "Bakery" en avant

Version	Mém.	Tps e.	Tps s.	Tps t
TREX$_1$	5.4	8.52	2.30	10.82
TREX$_2$	7.6	11.99	3.08	15.07

Notre outil nous permettant de pratiquer l'analyse en arrière, vérifions maintenant la propriété d'exclusion mutuelle en utilisant le principe du calcul des états en arrière. Une erreur que nous avons corrigée dans la dernière version de TREX ne nous permet pas de donner les résultats de l'analyse en arrière pour la version TREX$_1$. Dans ce cadre, nous ne tombons pas sur l'erreur d'Omega il est donc possible d'analyser l'exemple avec plus de deux processus.

La configuration initiale est la suivane (où n est le nombre de processus) :

$$((sc1, sc2, \cdots, scn), a1 \geq 0 \ \wedge \ a2 \geq 0 \ \wedge \ \cdots \ \wedge \ an \geq 0)$$

TAB. 7.4 – Statistique des performances pour l'algorithme du "Bakery" en arrière

nb proc.	TREX$_2$				nb. conf.
	Mém.	Tps e	Tps s	Tps t	
2	7.4	0.08	0.10	0.18	9
3	7.4	0.80	0.40	1.20	25
4	7.6	4.46	2.42	6.88	75

Dans le cadre de l'algorithme en arrière il n'y a pas besoin d'accélération pour cet exemple. Il suffit en effet de savoir si $c_1 < c_2$ ou $c_2 < c_1$ pour calculer l'ensemble des configurations atteignables. Nous pouvons vérifier que l'état initial n'est pas inclus dans une configuration atteinte.

7.3.3 Exclusion mutuelle : algorithme du ticket

L'algorithme du ticket est un exemple type dans la famille des algorithmes d'exclusion mutuelle, il est notamment présenté dans [And91, BGP99]. Dans le cas de l'algorithme du ticket, l'accès à la ressource commune est contrôlé par deux compteurs : un compteur d'entrée c_a (le ticket d'attente de la ressource) calcule le nombre de processus qui ont demandé l'accès à la ressource et un compteur de sortie c_s calculant le nombre de processus ayant relaché la ressource commune.

Chaque processus doit prendre un ticket avant d'entrer en section critique (moment où le processus utilise la ressource) et rendre le ticket en sortant. Il ne pourra utiliser cette ressource que si aucun autre processus n'est lui-même en train de l'utiliser, c'est-à-dire le numéro de son ticket est plus petit ou égal au nombre de processus ayant relaché la ressource. Examinons le cas où nous avons 2 processus. Pour plus de compréhension, chaque processus est modélisé par un automate étendu.

Dans cet exemple, l'automate $\mathcal{A}$ (produit asynchrone des deux processus) contient les données suivantes :

- $Q = \{repos_1,\ attente_1,\ SC_1\} \times \{repos_2,\ attente_2,\ SC_2\}$ où $repos$ est l'état par défaut dans lequel est le processus, $attente_i$ l'état lorsque le processus i a demandé à entrer dans la section critique et attend de pouvoir effectivement y rentrer et SC_i l'état dans lequel se trouve le processus i lorsqu'il a effectivement la ressource ($i \in \{1, 2\}$).
- $\mathcal{X} = \{id_1,\ id_2,\ c_a, c_s\}$ où
  - id_1 est le numéro du ticket courant pour le processus 1,
  - id_2 est le numéro du ticket courant pour le processus 2.
  - c_a est le compteur d'entrée.
  - c_s est le compteur de sortie.

  Toutes les variables sont des compteurs, il n'y a pas de paramètres.

La figure 7.6 représente la modélisation du processus i.

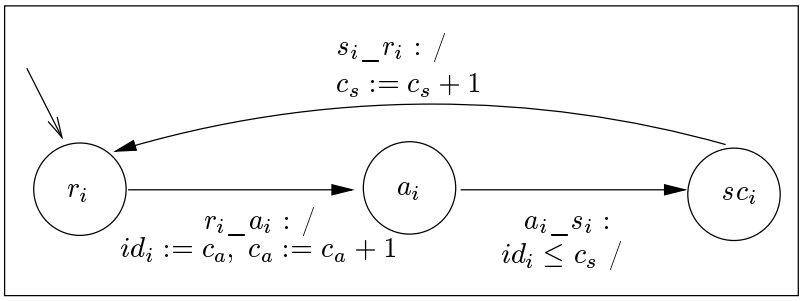


FIG. 7.6 – Le Ticket

Cet exemple permet une analyse en avant et en arrière utilisant la méthode d'accélé-

ration pour des compteurs.

TAB. 7.5 – Statistique des performances pour l'algorithme du ticket (1)

nb proc.	TREX_1				TREX_2				nb. conf.
(nb cpt)	Mém.	Tps e	Tps s	Tps t	Mém.	Tps e	Tps s	Tps t	
2(4)	8.4	7.65	4.07	11.72	8.9	5.19	6.23	11.42	38
3(5)	(*)				897	1057.33	6034.48	7091.81	219

(*) erreur dans l'exécutable.

Pour 4 processus, l'exécution avec TREX_2 a nécessité plus de $1,8$ Go de mémoire. Nous n'avons donc pas pu terminer cet exemple.

Vérifions maintenant la propriété d'exclusion mutuelle en utilisant le calcul en arrière des configurations.

Dans le cadre de l'analyse en arrière, la configuration de départ du calcul est la suivante (où n est le nombre de processus) :

$$((sc1, \cdots, scn), c_a \geq 0 \ \wedge \ c_s \geq 0 \ \wedge \ id_1 \geq 0 \ \wedge \ \cdots \ \wedge \ id_n \geq 0)$$

TAB. 7.6 – Statistique des performances pour l'algorithme du ticket en arrière

nb proc.	TREX_2				nb. conf.
(nb cpt)	Mém.	Tps e	Tps s	Tps t	
2(4)	7.6	5.31	0.90	6.21	10
3(5)	9.6	72.93	18.60	91.53	29
4(6)	33.5	8252.79	682.79	8935.04	84

Nous pouvons alors vérifier que l'état initial n'est pas inclus dans les configurations calculées par l'algorithme. Dans cet exemple, pour pouvoir terminer nous devons utiliser l'accélération en arrière, nous obtenons l'ensemble exact des configurations accessibles.

7.3.4 Exclusion mutuelle : algorithme de "Fischer"

Il existe plusieurs versions de l'algorithme de "Fischer", nous avons choisi la version décrite par [Lyn96].

Nous avons introduit cet exemple dans le chapitre 4 (page 77). Nous rappelons que ce modèle contient i processus, i horloges et 2 paramètres.

La figure 7.7 représente la modélisation du processus 1.

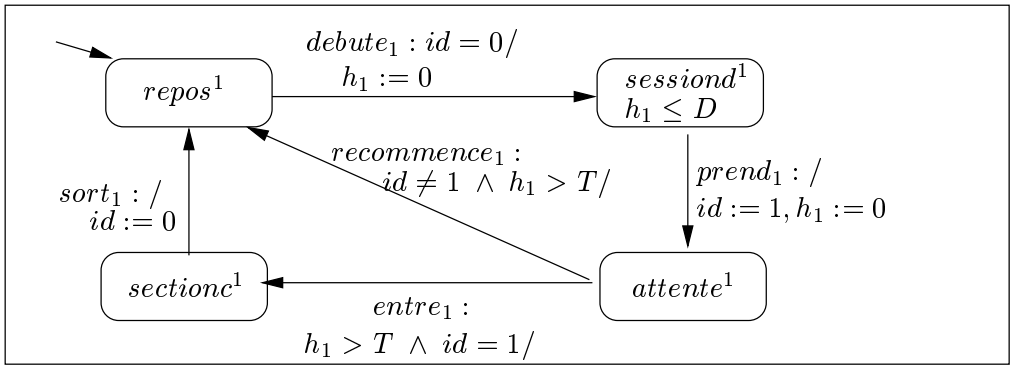


FIG. 7.7 – "Fischer" : Modélisation de processus 1

TAB. 7.7 – Statistique des performances pour l'algorithme de "Fischer" (1)

nb proc.	TREX_1				TREX_2			
	Mém.	Tps e	Tps s	Tps t	Mém.	Tps e	Tps s	Tps t
Instanciation = $T = 11 \ \wedge \ D = 7$								
2	(**)	-	-	-	7.8	0	0.18	0.18
3	(**)	-	-	-	9.1	0	6.41	6.41
4	(**)	-	-	-	104	0	935.76	935.76
contrainte initiale = $T > D \ \wedge \ D > 0$								
2	8.5	20.25	4.53	24.78	8.4	0	1.13	1.13
3					10.6	65.21	181.43	246.64
contrainte initiale = $T > 0 \ \wedge \ D > 0$ (synthèse)								
2					7.9	0	5.90	5.90

(**) Erreur dans l'exécutable.

TAB. 7.8 – Statistique des performances pour l'algorithme de "Fischer" (2)

nb proc.	TREX_3				nb. conf.
	Mém.	Tps e	Tps s	Tps t	
Instanciation = $T = 11 \ \wedge \ D = 7$					
2	-	-	-	-	29
3	-	-	-	-	279
4	-	-	-	-	4063
contrainte initiale = $T > D \ \wedge \ D > 0$					
2	8.4	0	0.87	0.87	39
3	10.6	0	170.82	170.82	369
contrainte initiale = $T > 0 \ \wedge \ D > 0$ (synthèse)					
2	7.9	0	4.41	4.41	61

Pour cet exemple, lorsque nous utilisons des paramètres, l'analyse utilisant REDUCE nous a donné des formules non linéaires, ce qui n'a pas été le cas avec notre élimination

de quantificateurs.

Nous avons analysé (sur une machine Linux avec 2GB de mémoire) l'exemple avec 4 processus lorsque la contrainte initiale est $T > D \ \wedge \ D > 0$ en utilisant notre élimination de quantificateurs. L'anlyse a demandé 800Mb de mémoire et a pris 38.4 heures.

Nous pouvons analyser cet exemple en appliquant le calcul des configurations en arrière.

TAB. 7.9 – Statistique des performances pour l'algorithme de "Fischer" en arrière

nb proc.	TREX_2				TREX_3				nb. conf.
	Mém.	Tps e	Tps s	Tps t	Mém.	Tps e	Tps s	Tps t	
$T = 11 \ \wedge \ D = 7$									
2	5.7	0	0.10	0.10	-	-	-	-	12
3	5.7	0	0.12	0.12	-	-	-	-	19
4	5.7	0	0.25	0.25	-	-	-	-	29
$T > D \ \wedge \ D > 0$									
2	7.4	0	0.40	0.40	7.5	0	0.30	0.30	12
3	7.5	0	0.75	0.75	7.5	0	0.59	0.59	19
4	7.5	0	1.48	1.48	7.5	0	1.30	1.30	29
$T > 0 \ \wedge \ D > 0$									
2	8	1.92	2.93	4.85	8	1.84	2.29	4.13	23
3	(*)				>1.8 Go				

(*) Nous obtenons des formules non linéaires.

Nous pouvons vérifier que pour chacun des exemples où les paramètres sont instanciés ou bien où nous avions donné la contrainte $T > D$, la configuration atteignable n'est pas incluse dans l'ensemble des configurations calculées. Par contre, dans les exemples sans contraintes particulières entre els paramètres T et D, nous obtenons dans l'ensemble des configurations calculées, la configuration d'état de contrôle $(repos_1, repos_2)$ (pour 2 processus) avec une contrainte sur les paramètres étant : $T > 0 \ \wedge \ T < 2D \ \wedge \ T < D \ \wedge \ D > 0$.

7.3.5 Protocole de Retransmission Bornée (un détail) — BRP (Bounded Retransmission Protocol)

Nous prenons ici un détail du protocole de retransmission bornée que nous avons introduit dans le chapitre sur l'analyse des systèmes à compteurs et à horloges paramétrés (section 4.5.3, page 79).

La figure 7.8 représente la réémission d'un message lors du fonctionnement du protocole de retransmission bornée. Nous avons deux horloges x et y représentant respectivement

les horloges de l'émetteur et du récepteur ainsi qu'un compteur c représentant le nombre de réémissions du message.

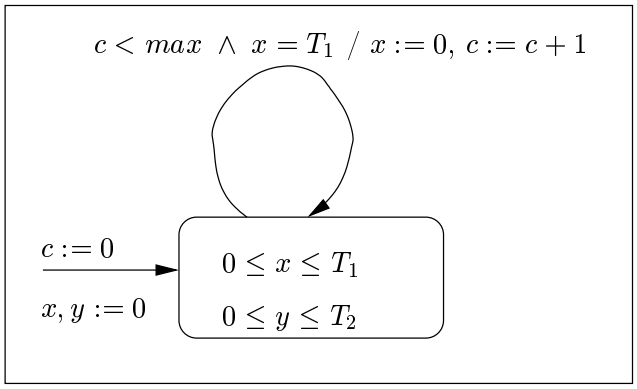


FIG. 7.8 – Détail du BRP

Nous avons analysé cet exemple en instanciant tous les paramètres BRP-v_c-v_{T_1}-v_{T_2} où $v_c \in \mathbb{N}$ est une valeur donnée au paramètre c et $v_{T_1}, v_{T_2} \in \mathbb{R}$ sont respectivement des valeurs données aux paramètres T_1 et T_2.

Nous analysons ainsi plusieurs cas de systèmes hétérogènes avec ou sans paramètres et dont le résultat et plus ou moins linéaire.

TAB. 7.10 – Statistique des performances pour un détail du BRP (1)

nb proc.	TREX_1				TREX_2			
	Mém.	Tps e	Tps s	Tps t	Mém.	Tps e	Tps s	Tps t
BRP-10-1-10	6.2	4.36	0.47	4.83	9.2	3.07	0.57	3.64
BRP-20-1-20	6.2	4.25	0.41	4.66	8.7	3.14	0.57	3.71
BRP-30-1-30	6.6	4.50	0.44	4.94	8.7	3.08	0.59	3.67
BRP-40-1-40	6.2	4.53	0.42	4.95	8.7	3.06	0.56	3.62
BRP-c-1-10	6.4	14.88	1.92	16.80	9.3	6.27	1.72	7.89
BRP-c-1-20	6.4	15.22	1.94	17.16	9.3	6.19	1.68	7.87
BRP-c-1-30	6.4	15.62	2.12	17.74	9.3	6.24	1.70	7.84
BRP-c-1-40	6.5	15.23	2.02	17.25	9.3	6.28	1.71	7.99
BRP-c-T_1-10	10.3	64.10	27.50	91.60	10.2	44.87	17.76	62.63
BRP-c-T_1-20	11.5	81.09	36.47	117.56	11	56.91	29.32	86.23
BRP-c-T_1-30	11.5	106.88	35.86	142.74	10.9	38.70	14.11	52.81
BRP-c-T_1-40	12.3	253.30	44.30	297.60	10.4	49.51	23.60	73.11
BRP-10-T_1-T_2	9.2	78.33	10.30	88.63	7.5	66.57	18.75	85.32
BRP-c-T_1-T_2	-	(**)	-	-	9.7	43.72	12.04	55.76

(**) Un beug dans l'exécutable TREX_1 ne permet pas de donner les résultats de cet exemple.

TAB. 7.11 – Statistique des performances pour un détail du BRP (2)

nb proc.	$TREX_3$				nb. conf.
	Mém.	Tps e	Tps s	Tps t	
BRP-10-1-10	-	-	-	-	5
BRP-20-1-20	-	-	-	-	5
BRP-30-1-30	-	-	-	-	5
BRP-40-1-40	-	-	-	-	5
BRP-c-1-10	-	-	-	-	5
BRP-c-1-20	-	-	-	-	5
BRP-c-1-30	-	-	-	-	5
BRP-c-1-40	-	-	-	-	5
BRP-c-T_1-10	10	21.17	68.57	89.74	14
BRP-c-T_1-20	10	21.28	68.21	89.49	14
BRP-c-T_1-30	10	20.95	67.70	88.65	14
BRP-c-T_1-40	10	21.67	68.19	89.86	14
BRP-10-T_1-T_2	8.2	16.79	154.06	170.85	15
BRP-c-T_1-T_2	9.8	14.93	29.63	44.56	7

Nous avons obtenu des contraintes non linéaires dans le cadre des versions BRP-c-T_1-y sans l'utilisation de notre élimination. Le fait d'utiliser notre package d'élimination de formules évite le problème des formules non linéaires et nous avons alors une analyse complètement automatique.

7.3.6 Protocole de retransmission bornée — Bounded Retransmission Protocol

Nous avons décrit ce protocole dans le chapitre 2 dans la section 2.4.4.2 (page 23).

7.3.7 Abstraction des compteurs et des horloges

Nous avons analysé le système abstrait ne contenant plus que les files d'attente décrit dans la section 3.2.2 (page 35).

L'analyse avec $TREX_1$ a pris 0.30s et 5.9Mo de mémoire, avec $TREX_2$ elle a pris 0.31 secondes et 6.4Mo de mémoire. Les valeurs sont sensiblement les mêmes car nous n'avons pas modifié les structures et les algorithmes des SRE entre ces deux versions de notre outil.

Nous avons alors obtenu l'ensemble des configurations atteignables représenté par le tableau 7.9. Le graphe symbolique où nous n'avons conservé que les messages envoyés au niveau application est représenté par la figure 7.10.

état	$abort$	$rtrans$	K	L
$s\_init \quad r\_init$	0	0	ϵ	ϵ
$s\_sendf \quad r\_init$	0	0	ϵ	ϵ
$s\_wa\_f \quad r\_init$	0	0	fst^*	ϵ
$s\_wa\_f \quad r\_wmf$	0	1	fst^*	fst^*
$s\_send0 \quad r\_wmf$	0	1	fst^*	fst^*
$s\_send0 \quad r\_wm0$	0	1	$m1^*$	$m1^*$
$s\_wa\_0 \quad r\_wmf$	0	1	fst^*m0^*	fst^*
$s\_wa\_0 \quad r\_wm0$	0	1	$m1^*m0^*$	$m1^*$
$s\_wa\_0 \quad r\_wm1$	0	1	$m0^*$	$m1^*m0^*$
	0	1	$m0^*$	fst^*m0^*
$s\_send1 \quad r\_wm1$	0	1	$m0^*$	$m0^*$
$s\_wa\_1 \quad r\_wm0$	0	1	$m1^*$	$m0^*m1^*$
$s\_wa\_1 \quad r\_wm1$	0	1	$m0^*m1^*$	$m0^*$
$s\_wa\_l \quad r\_init$	0	0	$last^*$	$m1^*last^*$
	0	0	$last^*$	$m0^*last^*$
$s\_wa\_l \quad r\_wmf$	0	1	fst^*last^*	fst^*
$s\_wa\_l \quad r\_wm0$	0	1	$m1^*last^*$	$m1^*$
$s\_wa\_l \quad r\_wm1$	0	1	$m0^*last^*$	$m0^*$
$s\_wa\_l \quad r\_end$	0	1	$last^*$	$m1^*last^*$
	0	1	$last^*$	$m0^*last^*$
$s\_wa\_rec \quad r\_init$	0	0	$last^*$	$last^*$
$s\_wa\_rec \quad r\_end$	0	1	$last^*$	$last^*$
$s\_error \quad r\_init$	1	0	ϵ	$m1^*$
	1	0	$last^*$	ϵ
	1	0	ϵ	$last^*$
	1	0	ϵ	$m0^*$
	1	0	ϵ	fst^*
	1	0	fst^*	ϵ
$s\_error \quad r\_wmf$	1	1	fst^*m0^*	fst^*
	1	1	fst^*last^*	fst^*
$s\_error \quad r\_wm0$	1	1	$m1^*m0^*$	$m1^*$
	1	1	$m1^*last^*$	$m1^*$
$s\_error \quad r\_wm1$	1	1	$m0^*m1^*$	$m0^*$
	1	1	$m0^*last^*$	$m0^*$
$s\_error \quad r\_end$	1	1	$last^*$	$last^*$

FIG. 7.9 – Ensemble des configurations atteignables du BRP (files d'attente)

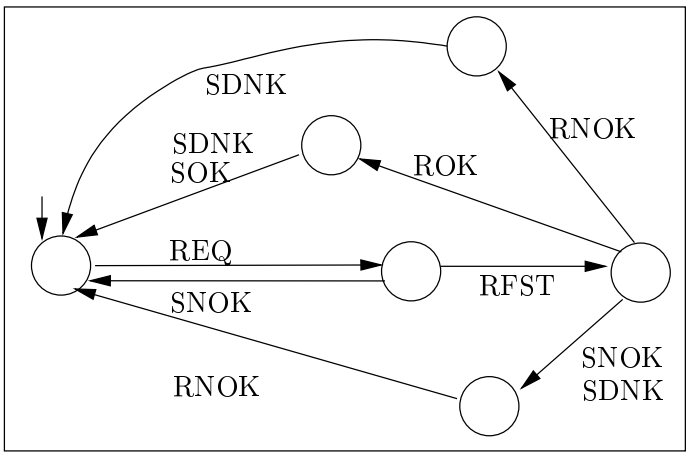


FIG. 7.10 – Graphe symbolique du BRP (files d'attente)

Ainsi, nous pouvons maintenant considérer l'ensemble des configurations atteignables comme invariant du modèle. Cet ensemble peut alors servir d'invariant pour calculer l'abstraction du modèle sans les files d'attente.

7.3.8 Abstraction du compteur de message i

Pour le moment, nous n'avons pas pu analyser complètement le système complet, contenant les deux files d'attente, les deux compteurs et les deux horloges par manque de mémoire sur nos machines. Par contre, nous avons réussi à analyser une version où nous avons abstrait le compteur i indiquant le nombre de messages du fichier envoyé. Nous supposons simplement savoir si le protocole est en train d'émettre le premier message, un message intermédiaire ou le dernier message. Nous analysons alors le système pour toutes les instances de N (le paramètre indiquant la longueur du fichier à retransmettre).

Les modèles utilisés pour l'émetteur et le récepteur sont représentés respectivement par les figures 7.11 et 7.12.

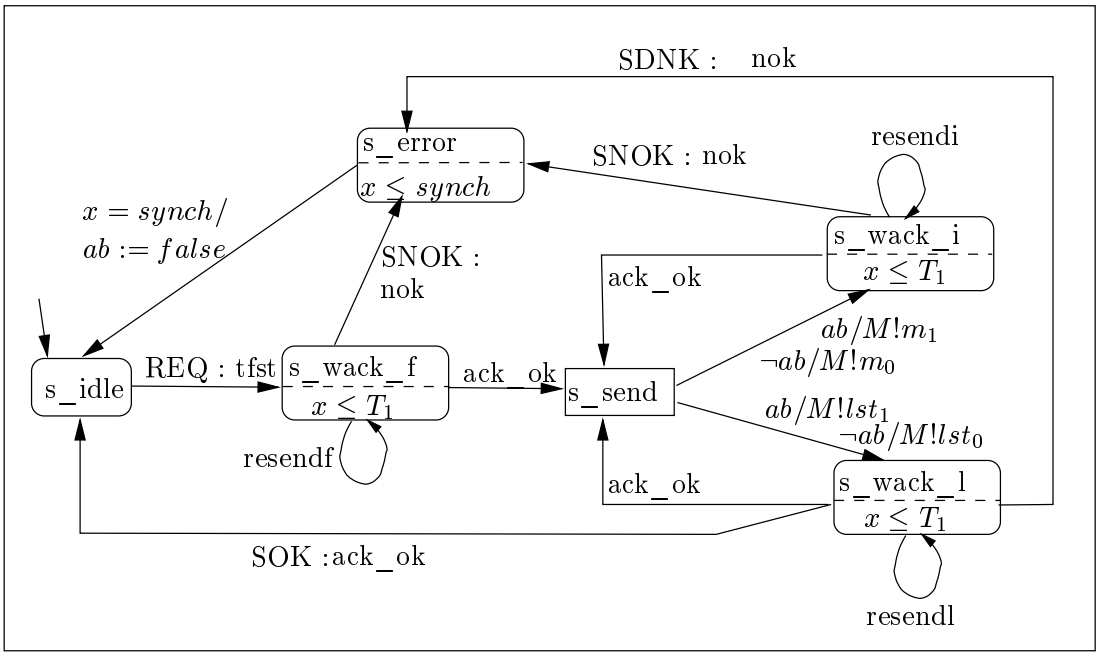


FIG. 7.11 – BRP — émetteur

$$tfst = \begin{array}{l} ab/x := 0,\ rc := 0,\ M!f_1 \\ \neg ab/x := 0,\ rc := 0,\ M!f_0 \end{array}$$

$$resendf = rc < Max\ \wedge\ x = T_1\ \wedge \left\{ \begin{array}{l} ab/M!f_1 \\ \neg ab/M!f_0 \end{array} \right\}, x := 0,\ rc := rc + 1$$

$$resendi = rc < Max\ \wedge\ x = T_1\ \wedge \left\{ \begin{array}{l} ab/M!m_1 \\ \neg ab/M!m_0 \end{array} \right\}, x := 0,\ rc := rc + 1$$

$$resendl = rc < Max\ \wedge\ x = T_1\ \wedge \left\{ \begin{array}{l} ab/M!l_1 \\ \neg ab/M!l_0 \end{array} \right\}, x := 0,\ rc := rc + 1$$

$$ack\_ok = x < T_1 \left\{ \begin{array}{l} ab/A?a_1 \\ \neg ab/A?a_0 \end{array} \right\},\ x := 0,\ rc := 0,\ ab := \neg ab$$

$$nok = rc = Max\ \wedge\ x = T_1 / x := 0$$

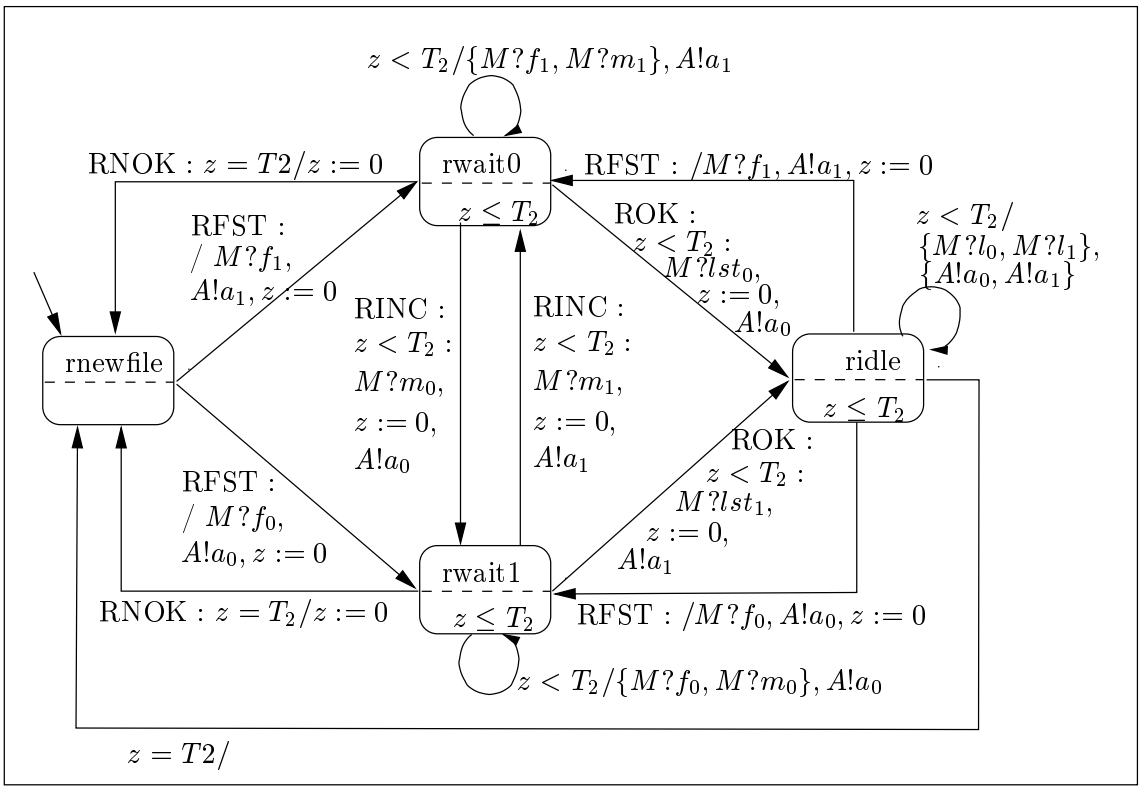


FIG. 7.12 – BRP — récepteur

7.4 Comparaison avec d'autres outils

A notre connaissance il n'existe pas d'outils de vérification permettant de manipuler à la fois plusieurs données complexes (files d'attente, horloges et compteurs non bornés) et des systèmes utilisant des paramètres.

Nous comparons dans cette section notre outil à une liste non exhaustive d'outils existants aussi bien pour les compteurs que pour horloges.

7.4.1 Les outils de comparaison

Les outils auxquels nous avons comparé TREX sont décrits ci-après. Nous les divisons en deux catégories. Tout d'abord les outils permettant d'analyser des systèmes à compteurs (le domaine de définition de ces compteurs pouvant être infini). Puis ceux manipulant des horloges, éventuellement des compteurs si le domaine de ces derniers est fini. Nous voyons que certains peuvent manipuler des paramètres.

Les outils sur les compteurs :

- DMC (pour Deductive Model Checking) [Del, DP99, DP00] est un vérificateur développé principalement par G. Delzanno permettant l'analyse et la vérification de propriétés de sûreté pour les systèmes à compteurs.

L'algorithme de calcul des états atteignables est basé sur l'utilisation de la programmation par logique contrainte (CLP : Constraint Logic Programming). Les ensembles de valuations des compteurs (ainsi que les systèmes à analyser) sont représentés par des clauses de la programmation logique. La vérification des propriétés se fait par une recherche en arrière des états. Cette recherche en arrière peut donner une sur-approximation des états. Par exemple, si nous considérons le système composé d'un seul état de contrôle sur lequel nous avons la transition $(q, 0 \leq x \leq T_1 \ / \ x := x+2)$. Si nous commençons l'analyse avec l'ensemble $x \geq 0 \ \wedge \ T_1 \geq 0$, l'ensemble des configurations atteintes est alors : $x \geq 0 \ \wedge \ T_1 \geq 0$ qui est une sur-approximation.
De plus, une méthode d'élargissement combinée à la relaxation des entiers vers les réels a été implantée dans l'outil pour permettre l'accélération du calcul dans certains cas. Le résultat donné (ensembles des configurations) est alors, ici aussi, une sur-approximation.
Il est possible de manipuler des contraintes (pour cela, il faut déclarer des variables qui ne sont jamais affectées). Les contraintes sur ces paramètres doivent rester linéaires. L'utilisation de contraintes peut introduire des comportements infinis. Par exemple, dans l'exemple précédent si l'ensemble de départ est $x = T_2 \ \wedge \ T_2 \geq 0 \ \wedge \ T_1 \geq 0$, le calcul de l'ensemble des états ne va pas s'arrêter. Si nous utilisons TReX l'analyse s'arrête et nous obtenons alors l'ensemble suivant : $x = -2+T_2-2n_0 \wedge T_1 \geq 0 \wedge -T_1+T_2 \leq 2 \wedge T_2 \geq 2 \wedge -T_2+2n_0 \leq -2 \wedge n_0 \geq 0$.

- Lash (pour Liège Automata-based Symbolic Handler) [las] est un outil permettant l'analyse de systèmes à compteurs ou les systèmes à files d'attente. Les valuations des compteurs sont représentées par NDD (Numerical Decision Diagram [Boi98]) et les contenus des files d'attente (celles-ci étant parfaites) par des QDD (Queue Decision Diagram [BG96, BGWW97]). Pour les compteurs la notion d'accélération des ensembles périodiques [BW94] a été implantée, mais c'est à l'utilisateur de donner le circuit par lequel il aimerait accélérer. De plus, dans la version actuelle de l'outil, il n'est pas possible de visualiser l'ensemble des configurations atteintes, ce qui ne permet pas de vérifier si l'ensemble obtenu est comparable à celui obtenu par TReX.
  Il est possible d'utiliser des paramètres (définition de variables sans affectation) si l'effet des circuits reste NDD-représentable (notamment il n'est pas possible de manipuler des formules non linéaires). L'usage des paramètres peut parfois empêcher l'emploi de l'accélération. Reprenons un exemple composé d'un seul état de contrôle et d'une transition $(q, \ /x := x+T_2, q)$ où nous n'avons pas de garde et où la variable x est incrémentée de T_2 à chaque passage. Si nous instancions le paramètre à une valeur fixe (par exemple 2) mais que nous laissons le paramètre dans le système, Lash ne s'arrête pas car l'effet du circuit : $n * T_2$ n'est pas NDD-représentable.
- Composite [YKTB01, Bul00] est un outil de vérification pour les systèmes à compteurs manipulant des variables booléennes. Cet outil combine différentes représentations symboliques telles que les BDD pour représenter les formules de logique booléenne et une représentation à base de polyhèdres pour représenter les formules d'arithmétique linéaire.
  Une méthode d'élargissement a été développée pour accélérer le calcul des états

atteignables. Cette méthode donne alors une sur-approximation de l'ensemble des états atteignables. Il est possible de faire un calcul en avant ou arrière. Le calcul en arrière donne éventuellement une sur-approximation. Pourtant, nous n'avons pu utiliser le calcul en avant que dans un seul cas (exemple du "Bakery" avec deux processus où le résultat avait été trouvé en 2,36s) dans les autres cas où nous avons essayé l'algorithme n'était pas fini après avoir utilisé plus de 500Mb de mémoire. Nous comparons donc seulement les performances en arrière.
Il est possible de déclarer des paramètres, les contraintes faisant intervenir ces paramètres doivent rester linéaires.

Les outils manipulant des horloges :

- UPPAAL [LPY97, LP00b, LP00a] est développé par W. Yi, K. Larsen et P. Petterson. Cet outil permet l'analyse de systèmes temporisés pouvant communiquer deux à deux et pouvant manipuler des variables entières bornées. La version courante de l'outil ne permet pas l'analyse de systèmes paramétrés. Il n'est pas possible d'obtenir l'ensemble des configurations atteignables. Nous pouvons faire une simulation interactive de l'analyse d'un système ou demander la vérification d'une propriété de logique temporelle. La représentation des valuations est faite par des CDD (Constraint Decision Diagram) [LWYP98, BLP+99].
  Une extension récente de UPPAAL [HRSV01] — non disponible pour le moment — permet la vérification d'une classe restreinte de systèmes temporisés paramétrés. Ce sont les systèmes dits "L/U Automata". Les systèmes analysés sont des systèmes temporisés où il est possible de comparer des horloges à des paramètres $x-y\{<,\geq,>,\leq\}c_0+c_1.l_1...$ où c_i sont des constantes et l_i des paramètres. Il existe deux catégories de paramètres les *plus grands* sont ceux qui apparaissent dans des formules faisant intervenir les signes $\{>,\geq\}$ et les *les plus petits* ceux faisant intervenir les signes $\{<,\leq\}$. Il n'est donc à priori pas possible de déclarer $x = P1$ car chaque paramètres ne doit appartenir qu'à un seul ensemble. De plus, l'analyse de ces systèmes n'est pas faite avec tous les paramètres, [HRSV01] ont en effet développé des techniques d'abstraction permettant de réduire le nombre de ces paramètres en choisissant pour certains des valeurs induites par l'abstraction. Enfin, l'analyse est menée sans technique d'accélération, les ensembles de valuations des horloges étant représentés par une extension des matrices de bornes. Dans cette version il n'est pas possible de manipuler des formules non linéaires.
- KRONOS [DOTY96, Yov97] est un outil de vérification de systèmes temporisés développé à Vérimag principalement par S. Yovine et S. Tripakis. Kronos ne permet pas l'analyse de systèmes contenant des paramètres. La représentation des valuations des horloges est exprimée par des matrices à différence de bornes (DBM : Difference Bounded Matrix). Plusieurs abstractions [DT98] (simulation, extrapolation, inclusion, activité, couverture convexe) ont été développées pour aider à la terminaison du calcul des états atteignables. Les 4 premières abstractions ne sont pas exactes au sens où elles ne calculent pas l'ensemble exact des configurations atteignables mais restent sauves pour la vérification de propriétés. L'abstraction de la couverture convexe permet quant à elle de réduire la taille de l'ensemble des configurations

calculées mais donne un sur-ensemble.

- HYTECH [HH95, HHWT95, AHH96] est un outil permettant l'analyse d'automates hybrides linéaires. Les automates temporisés sont un sous-ensemble des automates hybrides où la dérivée des horloges est constante. HYTECH permet la manipulation de paramètres. Les valuations des horloges sont représentées par une bibliothèque implantant des polyhèdres. Il est possible de faire une analyse en avant ou une analyse en arrière.

Pour ces exemples nous avons utilisé les outils sous les versions suivantes : LASH-0.8, COMPOSITE-0.1, UPPAAL-3.1.64, KRONOS-2.4.4 et HYTECH-1.04.

7.4.2 Les comparaisons sur les compteurs

Nous commençons avec l'exemple de l'ascenseur. Les lignes contenant une étoile (*) sont l'analyse de l'exemple donné sans avoir recours à une méthode d'accélération.

TAB. 7.12 – Comparaison pour les compteurs en avant (1)

Exemple	TREX		LASH		
	Mém.	Tps	Mém.	Tps	Comm.
Ascenceur (N instancié)					
Asc2 (*)	8.1	0.11	1.4	0.51	8 pas.
Asc2	8.2	0.14	0.10	1.95	4 pas.
Asc3 (*)	8.1	0.20	1.4	0.73	
Asc3	8.2	8.95	0.10	2.11	4 pas.
Asc4(*)	8.1	0.24	1.4	0.96	
Asc4	8.2	14.18	0.11	2.42	4 pas.
Asc5(*)	8.1	0.43	1.4	1.27	
Asc5	8.2	12.01	0.11	2.58	4 pas.
Asc6(*)	8	0.56	1.4	1.48	
Asc6	8.2	11.86	0.12	2.69	
Asc7(*)	8	0.71	1.4	1.83	
Asc7	8.2	12.01	0.18	2.89	
Asc8(*)	8	0.92	1.4	2.08	
Asc8	8.2	11.85	0.21	3.27	
Asc9(*)	8	1.08	1.4	2.39	
Asc9	8.2	11.79	0.23	3.38	
Asc10(*)	8	1.25	1.4	2.69	
Asc10	8.2	11.87	0.24	3.47	
Asc100(*)	11.3	83.62	1.6	42.23	
Asc100	8.2	11.99	0.57	6.27	
Asc1000(*)	$\perp$		3.8	627.49	6994 conf.
Asc1000	8.2	12.16	0.85	9.41	
Asc10000	8.2	11.89	1.17	13.51	

TAB. 7.13 – Comparaison pour les compteurs en avant (2)

Exemple	TReX		Lash		
	Mém.	Tps	Mém.	Tps	Comm.
Ascenceur (N non instancié)					
AscN	8.3	13.11			pas d'analyse
Asc∞	8.1	6.41	0.24	2.85	
Ticket					
Tick2	8.9	11.42	1.9	21.33	8 pas.
Tick3	897	7091.81	14.4	339.07	12 pas.
"Bakery"					
Bak2	7.6	15.07	0.11	2.37	
Bak3			(**)		

(**) Le problème avec Lash est de donner les bons circuits permettant à l'analyse de s'arrêter. Comme il n'est pas possible de visualiser une trace partielle de l'avancée de l'analyse, l'utilisateur doit alors refaire à la main l'analyse pour trouver les bons circuits. Dans le cas de deux processus, nous avons utilisé les circuits que TReX avait utilisé.

Ainsi, les exemples traités avec Lash prennent moins de temps et d'espace qu'avec TReX. Nous pensons que cela est principalement dû au fait que nous ne donnons pas à priori les circuits par lesquels l'algorithme doit accélérer ce qui permet de ne pas obliger l'utilisateur à avancer à la main l'analyse pour trouver les circuits. Mais d'un autre côté, à chaque nouvelle configuration atteinte nous testons la possibilité d'accélérer. Dans cet exemple-ci cela peut prendre beaucoup de temps car il n'est pas possible d'accélérer par des boucles simples, il faut impérativement remonter à des boucles de niveau 2.

TAB. 7.14 – Comparaison pour les compteurs en arrière

Exemple	TReX		DMC			Composite		
	Mém.	Tps	Mém.	Tps	Comm.	Mém.	Tps	Comm.
"Bakery"								
Bak2	7.4	0.18	8.6	0	SA	9.8	1.02	SA
Bak3	7.4	1.20	8.6	0.1	SA	14.1	5.30	SA
Bak4	7.4	6.88	8.6	3.3	SA	20.8	12.45	SA
Ticket								
Tick2	7.6	6.21	8.6	0.7	A	11.9	2.19	A
Tick3	9.6	91.53	8.6	95.6	A	19.2	24	A
Tick4	33.5	8935.04	9.3	33791.1	A	38.6	257.84	A

Au travers de l'exemple du "Bakery" (sans besoin d'accélération), nous observons que TReX permet d'analyser les systèmes dans un temps de même ordre de grandeur que DMC. Par contre, au sujet de l'exemple du ticket nous voyons TReX reste dans le même ordre de grandeur que DMC bien que ce dernier donne un ensemble sur-approximé. Com-

POSITE quant à lui est plus rapide pour l'exemple du ticket mais donne un ensemble sur-approximé ce que ne fait pas TREX.

7.4.3 Les comparaisons sur les horloges

Regardons maintenant les comparaisons pour les automates temporisés. Nous commençons par le cas de l'algorithme du "Fisher".

TAB. 7.15 – Comparaison pour les horloges

Exemple	TREX		UPPAAL		KRONOS		HYTECH	
	Mém.	Tps	Mém.	Tps	Mém.	Tps	Mém.	Tps
"Fischer" en avant								
Fisch2-11-7	7.8	0.18	1	0.2	1.4	0.02	5.9	0.17
Fisch3-11-7	9.1	6.41	1	0.2	1.8	0.23	6.7	1.57
Fisch4-11-7	104	935.76	1	0.5	3.1	0.75	145.9	138.51
Fisch2-T>D	8.4	0.87	-	-	-	-	5.8	0.25
Fisch3-T>D	10.6	170.82	-	-	-	-	9.6	2.79
Fisch2-T-D	7.9	4.41	?(*)	2	-	-	5.8	0.40
"Fischer" en arrière								
Fisch2-11-7	5.7	0.10			1.4	0	5.9	0.09
Fisch3-11-7	5.7	0.12			1.6	0.02	6.5	0.85
Fisch4-11-7	5.7	0.25			2.9	0.25	22.5	11.67
Fisch2-T>D	7.5	0.30	-	-	-	-	5.9	0.28
Fisch3-T>D	7.5	0.59	-	-	-	-	8.2	1.74
Fisch4-T>D	7.5	1.30	-	-	-	-	60.2	74.02
Fisch2-T-D	8	4.13			-	-	2.9	0.23

(*) Les résultats de Uppaal avec des paramètres sont des résultats que nous avons repris de l'article [HRSV01]. La version du fischer utilisée est celle décrite par [AL91, AHV93, Tri98] où seul le paramètre T a été gardé non instancié.

Dans cet algorithme, il n'y a pas besoin de l'accélération, les performances en temps de TREX (notamment comparés à HYTECH) sont assez défavorables. Nous pensons que cela peut provenir du fait que notre algorithme essaie d'accélérer à toutes les itérations ce qui le ralentit considérablement dans ce cas. Nous notons qu'il existe une plus grande différence dans le cas de l'analyse en avant que lors de l'analyse en arrière.

Les résultats pour le détail du protocole de retransmission borné sont donnés dans le tableau 7.16. Pour cet exemple, TREX est le seul à pouvoir analyser tout les cas, car il n'existe pas d'algorithme d'accélération pour les autres outils. En effet, lorsque nous utilisons des paramètres il devient alors impossible de calculer la borne maximale de comparaison pour chacune des horloges.

TAB. 7.16 – Comparaison pour les horloges

Exemple	TReX		Uppaal		Kronos		Hytech	
	Mém.	Tps	Mém.	Tps	Mém.	Tps	Mém.	Tps
Détail du BRP								
BRP-10-1-10	9.2	3.64	1	0.2	1.5	0.1	5.7	0.19
BRP-20-1-20	8.7	3.71	1	0.2	1.5	0.35	5.7	0.24
BRP-30-1-30	8.7	3.67	1	0.3	1.5	0.36	5.7	0.31
BRP-40-1-40	8.7	3.62	1	0.3	1.6	0.34	5.7	0.41
BRP-c-1-10	9.3	7.89	-	-	-	-	5.8	0.24
BRP-c-1-20	9.3	7.87	-	-	-	-	5.8	0.45
BRP-c-1-30	9.3	7.84	-	-	-	-	5.9	0.80
BRP-c-1-40	9.3	7.99	-	-	-	-	5.9	1.09
BRP-c-T1-10	10.2	62.63	-	-	-	-	-	-
BRP-c-T1-20	11	86.23	-	-	-	-	-	-
BRP-c-T1-30	10.9	52.81	-	-	-	-	-	-
BRP-c-T1-40	10.4	73.11	-	-	-	-	-	-
BRP-10-T1-T2	7.5	85.32	-	-	-	-	-	-
BRP-c-T1-T2	9.8	44.56	-	-	-	-	-	-

7.4.4 Conclusion

Le tableau 7.17 synthétise les possibilités de chacun des outils que nous avons considérés dans ce comparatif. Nous voyons que TReX est le seul a pouvoir traiter à la fois des compteurs non bornés, des horloges et des files d'attente. De plus TReX permet l'analyse de systèmes contenant des paramètres.

TAB. 7.17 – Récapitulatif comparaison

Nom	Compt.	Horl.	Files	Analyse		Accél.	Param.
				Avant	Arrière		
DMC	×	-	-	-	×	NE	PL
Lash	×	-	×	×	-	E	PL
Composite	×	-	-	×	×	NE	PL
Uppaal	× (bornés)	×	×	×	×	-	-
P-Uppaal	× (bornés)	×	-	×	-	-	PL
Kronos	-	×	-	×	×	-	-
Hytech	× (bornés)	×	-	×	×	-	PL
TReX	×	×	×	×	×	E	PNL

En conclusion, il est vrai que TReX est moins performant pour les systèmes à compteurs ou à horloges sans paramètres car les algorithmes de notre prototype sont développés pour un cadre de système plus grand. Il reste encore beaucoup d'améliorations à apporter à TReX (par exemple éviter de chercher des circuits pour accélérer à partir de tous les

états de contrôle) mais il nous a montré que les méthodes décrites permettent d'analyser des systèmes non traités jusqu'à présent.

Chapitre 8

Conclusion

Résumé et bilan

Dans le cadre des technologies de la télécommunication, les entreprises développent des protocoles dits *de communication* pour gérer l'envoi et la réception de données entre machines. Ces protocoles fonctionnent sur le principe d'envoi de messages entre deux parties par l'intermédiaire de canaux non fiables (i.e. pouvant perdre des messages). Pour être certain que tous les messages ont bien été reçus, les techniques généralement employées consistent à réémettre les messages potentiellement perdus (le nombre de réémissions peut être fixe ou non borné), et/ou à attendre un laps de temps déterminé (au moins le temps physique de transmission entre les deux parties) avant de conclure que la transmission d'un message a échoué (souvent l'échec est déclaré lorsque l'une des deux parties ne reçoit pas d'acquittement pour le message qu'elle vient d'envoyer). De plus, les systèmes sont souvent modélisés en fonction de paramètres (par exemple, le temps de transmission entre les machines n'est pas forcément une valeur fixe).

Nous avons donc travaillé sur un modèle mathématique permettant la vérification de spécifications (comportement attendu des systèmes) pour des protocoles manipulant à la fois des compteurs, des files d'attente ou des horloges, ces variables pouvant contenir des valeurs liées aux paramètres. Le but de l'analyse est de calculer l'ensemble des comportements possibles du système puis de vérifier qu'aucun de ces comportements ne viole une spécification attendue pour le système. Le problème ici est que l'ensemble des comportements peut être infini. En effet, un comportement est fonction des valeurs que peuvent prendre les variables du système au fil de l'exécution et dans notre cas certaines variables ont un domaine de définition infini (par exemple les compteurs). Il est donc nécessaire de développer tout d'abord des représentations de comportements permettant de représenter de manière finie un ensemble infini et ensuite des méthodes permettant de calculer de manière finie un ensemble de comportements infinis.

Plus formellement, nous nous sommes placés dans le cadre de l'analyse automatique des systèmes (*model-checking*) et nous avons développé une technique automatique pour l'analyse de systèmes complexes. La première étape de cette analyse consistant éventuellement à abstraire le modèle considéré pour obtenir un modèle plus simple sur lequel nous pouvons alors travailler. Ayant obtenu ce modèle, nous avons montré qu'il nous fallait alors pouvoir représenter les configurations du système et pouvoir appliquer la notion d'accélération en manipulant des objets de cette représentation. Pour notre part, nous avons défini une représentation, les matrices de bornes paramétrées contraintes, permet-

tant de manipuler à la fois des valuations de variables réelles (les horloges) et de variables entières (les compteurs) faisant intervenir des paramètres. Nous avons défini sur cette représentation une méthode exacte d'accélération qui nous permet dans la plupart des cas de donner l'ensemble exact des configurations atteignables du système considéré.

Du côté pratique, nous avons implanté ces méthodes dans un outil TReX. Cet outil est à notre connaissance le seul pouvant manipuler en même temps et de manière exacte des compteurs, des horloges et des files d'attente. Bien que ce ne soit qu'un prototype nous avons pu vérifier des exemples conséquents tels que le protocole de retransmission bornée ou le protocole de routage "Root Contention IEEE 1394".

Perspectives

D'un point de vue pratique, à court terme, nous aimerions avancer dans deux directions.

Tout d'abord, nous voulons tout d'abord améliorer TReX pour que nous puissions le tester sur d'autres exemples concrets de protocoles existants. Nous pensons qu'il est possible de simplifier encore notre représentation des termes arithmétiques et/ou d'utiliser d'autres procédures de décision plus efficace pour résoudre les problèmes de décision des formules arithmétiques. Nous voudrions aussi améliorer les représentations des matrices de bornes contraintes en se basant sur les méthodes étudiées pour les matrices de bornes sans paramètres (par exemple, ne représenter dans notre structure que les bornes utiles, les autres pouvant alors être déduites). La seconde direction est l'implantation des techniques de la vérification de propriétés de "fairness" pour avoir un prototype complet et automatique de vérification d'automates étendus.

D'un point de vue théorique, une première idée est de chercher si d'autres représentations, plus compactes, telles que les DDD ou les CDD peuvent être étendues aux paramètres et à l'accélération.

Pour ce travail, nous avons choisi de considérer les problèmes de paramétrisation dans le sens où les variables peuvent être comparées et affectées à des paramètres (problème de la synthèse de contrainte sur les paramètres). Il existe un autre problème lié à la paramétrisation qui est dû aux nombres de processus mis en jeu. En effet, [GS92, KMM+97, KP98, ABJN99, JN00, BJNT00, BMT01] posent et traitent le problème de la vérification de systèmes lorsque l'on considère un nombre non borné (paramétré) de processus. Ces travaux considèrent un ensemble de processus qui ne sont différents que par leur identifiant. Une recherche intéressante serait de combiner les méthodes décrites par [KMM+97, BJNT00] et les méthodes que nous avons développées pour pouvoir ainsi analyser des systèmes dans leur généralité (sans restriction du nombre de processus et sans restriction sur les données).

Un problème crucial à l'analyse des protocoles de communication récents est le problème de la mobilité. Elle impose des changements dynamiques de la structure logique des réseaux de communication. En effet, les processus peuvent à tout moment être créés, puis communiquer avec les autres processus et enfin disparaître. Il faut donc pouvoir considérer des familles de processus, ce qui nous ramène au problème de la paramétrisation due au nombre de processus.

Bibliographie

[AAB+99a] P. Abdulla, A. Annichini, S. Bensalem, A. Bouajjani, P. Habermehl, and Y. Lakhnech. Verification of infinite-state systems by combining abstraction and reachability analysis. In *11th International Conference on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 146–159, July 1999.

[AAB99b] P. Abdulla, A. Annichini, and A. Bouajjani. Symbolic verification of lossy channel systems : Application to the bounded retransmission protocol. *Lecture Notes in Computer Science*, 1579 :208–222, 1999.

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *12th International Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 419–434, July 2000.

[ABB01] P. Abdulla, L. Boasson, and A. Bouajjani. Effective lossy queue languages. In *Proc. 28th Coll. on Automata Languages and Programming, ICALP'01*, Lecture Notes in Computer Science, Crete, Greece, July 2001. Springer-Verlag.

[ABJ98] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. *Lecture Notes in Computer Science*, 1427 :305–317, 1998.

[ABJN99] P. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. *Lecture Notes in Computer Science*, 1633 :134–150, 1999.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model Checking for Real-Time Systems. In *Proc. 5^{th} IEEE Int. Symp. on Logic in Computer Science, Philadelphia*, pages 414–425, 1990.

[ACD+92] R. Alur, C. Coucourbetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification on automata emptiness. *RTSS'92. IEEE*, 1992.

[ACM84] D.S. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition i : the basic algorithm. In *SIAM Journal on Computing*, volume 13(4), nov 1984.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126 :183–235, 1994.

[AH92] R. Alur and T. Henzinger. Logics and models of real-time. In J.W. de Bakker, K. Huizing, W.P. de Roover, and G. Rozenberg, editors, *Real Time : Theory*

in Practice, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer-Verlag, 1992.

[AHH96] R. Alur, T. A. Henzinger, and P-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22 :181–201, 1996.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.

[AHV93] R. Alur, T. Henzinger, and M. Vardi. Parametric real-time reasoning. In *Proc. 25th Annual Symposium on Theory of Computing*, ACM Press, pages 592–601, 1993.

[AJ93] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proc. 8^{th} annual, Montreal Canada*, pages 160–170, jun 1993.

[AJ94] P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Lecture Notes in Computer Science*, 820 :316–327, 1994.

[AL91] M. Abadi and L. Lamport. An old-fashioned recipe for real-time. *Lecture Notes in Computer Science*, 600 :1–27, 1991.

[And91] G. R. Andrews. *Concurrent Programming : Principles and Practice*. The Benjamin/Cummings Publishing Company, 1991.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants*. Collection Etudes et Recherche en Informatique, masson edition, 1992.

[BBF+00] S. Bensalem, M. Bozga, J-C. Fernandez, L. Ghirvu, and Y. Lakhnech. A transformational approch for generating non-linear invariants. In *Proceedings of Static Analysis Symposium SAS'00*, volume 1824 of *Lecture Note Computer Science*. Springer-Verlag, 2000.

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems (extended abstract). In *Proc. 9th International Computer Aided Verification Conference*, Lecture Notes in Computer Science, pages 167–178, 1997.

[BCALS01] A. Bouajjani, A. Collomb-Annichini, Y. Lakhnech, and M. Sighireanu. Analysing fair parametric extended automata. In *The 8th International Static Analysis Symposium, SAS'01*, Lecture Notes in Computer Science, pages 335–355, July 2001.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking : 10^{20} states and beyond. In *Proc. 5^{th} LCS*, pages 414–425, Philadelphia, 1990.

[BFG+99a] M. Bozga, J-C. Fernandez, L. Ghirvu, S. Graf, J-P. Krimm, and L. Mounier. If : An intermediate representation and validation environment for timed asynchronous systems. In *FM'99*, volume 1708 of *Lecture Notes in Computer Science*, pages 307–327. Springer, Sep 1999.

[BFG+99b] M. Bozga, J-C. Fernandez, L. Ghirvu, S. Graf, J-P. Krimm, L. Mounier, and J. Sifakis. If : An intermediate representation for sdl and its applications. In *SDL FORUM'99*, pages 423–440. Elsevier, June 1999.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. *Lecture Notes in Computer Science*, 1102 :1–12, 1996.

[BGMS98] M. Bozga, S. Graf, L. Mounier, and J. Sifakis. The intermediate representation if. Technical report, Verimag, 1998.

[BGP99] T. Bultan, R. Gerber, and William Pugh. Model-checking concurrent systems with unbounded integer variables : symbolic representation, approximation and experimental results. In *ACM Transactions on Programming Languages and Systems*, volume 21 :4, pages 747–789, July 1999.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proceedings SAS'97*, September 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic Reachability Analysis of FIFO Channel Systems with Nonregular Sets of Configurations (extended abstract). In *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *Lecture Notes in Computer Science*, pages 560–570, Bologna, Italy, July 1997. Springer-Verlag.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *12th International Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418, July 2000.

[BJW01] B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Automated Reasoning, First International Joint Conference, IJCAR'01*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625, 2001.

[BLO98] S. Bensalem, Y. Lakhnech, and S. Owre. Computing Abstractions of Infinite State Systems Compositionally and Automatically. In A. Hu and M. Vardi, editors, *Proceedings of CAV'98 (Vancouver, Canada)*, volume 1427 of *LNCS*, pages 319–331. Springer, June 1998.

[BLP+99] G. Behrmann, K.G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *proceedings of the 11th International Conference on Computer Aided Verification, CAV'99*, number 1633 in Lecture Notes in Computer Sciences, pages 341–353, Jul 1999.

[BLS00] K. Baukus, Y. Lakhnech, and K. Stahl. Verifying universal properties of parameterized networks. In *Proceedings of FTRTFT*, 2000.

[BMT01] A. Bouajjani, A. Muscholl, and T. Touili. Permutation rewriting and algorithmic verification. In *16th IEEE Symp. on Logic in Computer Science, LICS'01*, Lecture Notes in Computer Science, page ? ?, July 2001.

[Boi98] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, Université de Liège, 1998.

[Bou01] A. Bouajjani. Languages, rewriting systems, and verification of infinite-state systems. In *Proc. 28th Coll. on Automata Languages and Programming, ICALP'01*, Lecture Notes in Computer Science, Crete, Greece, July 2001. Springer-Verlag.

[BRW98] B. Boigelot, S. Rassart, and P. Wolper. On the expressiveness of real and integer arithmetic automata. *Lecture Notes in Computer Science*, 1443 :152–? ?, 1998.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing surveys*, 24(3) :293–318, 1992.

[Bul98] T. Bultan. *Automated Symbolic Analysis of Reactive Systems*. PhD thesis, Université de Maryland, 1998.

[Bul00] T. Bultan. Action language : A specification language for model checking reactive systems. In *Proc. 22nd International Conference on Software Engineering, ICSE 2000*, pages 335–344, June 2000.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proceedings of the 6th International Conference on Computer Aided Verification*, volume 818 of *Lecture Note Computer Science*, pages 55–67, 1994.

[BW98] B. Boigelot and P. Wolper. Verifying systems with infinite but regular state spaces. In *Proceedings of the 10th International Conference on Computer Aided Verification*, volume 1427 of *Lecture Note Computer Science*, pages 88–95, 1998.

[BZ83] D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2) :323–342, April 1983.

[CAS01] A. Collomb-Annichini and M. Sighireanu. Parameterized reachability analysis of the ieee 1394 root contention protocol using trex. In *Proceedings of RT-TOOLS'01*, page ? ?, 2001.

[CBM89] O. Coudert, C. Berthet, and J.C. Madre. Varification of synchronous sequential machines based on symbolic execution. In *International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Note in Computer Science*, 1989.

[CC77] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages*, 1977.

[Céc98] Gérard Cécé. *Vérification, analyse et approximations symboliques des automates communicants*. PhD thesis, Ecole Normale Supérieure de Cachan, Janvier 1998.

[Céc99] G. Cécé. Ring networks are easier to verify. Technical report, Université de Liège, Belgique, 1999.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, May 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification. *ACM Trans. on Programming Languages and Systems*, 8(2) :244–263, April 1986.

[CF87] A. Choquet and A. Finkel. Simulation of linear fifo nets having a structured set of terminal markings. In *Proc. 8^{t}h European Workshop on Applications and Theory of Petri Nets*, pages 95–112, 1987.

[CFP96] G. Cécé, A. Finkel, and S. Purushotaman. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(3) :20–31, 1996.

[CGL94] E.M. Clarke, O. Grumberg, and D.E. Long. Model-checking and abstraction. *ACM Transactions on Prgramming Languages and Systems*, 16(5), 1994.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL'78 ACM*, 1978.

[CH88] T. Coquand and G. Huet. The calculus of construction. *Information and Computation*, 76(2), 1988.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata safety analysis and presburger arithmetic. Technical Report LSV-98-1, ENS de Cachan, France, mar 1998.

[CJ99] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proceedings of the 10th International Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer-Verlag, sep 1999.

[CJM00] E. M. Clarke, S. Jha, and W. Marrero. Partial order reductions for security protocol verification. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, 2000.

[CS01] M. Colon and H. Sipma. Synthesis of linear ranking functions. In *Proceedings of TACAS'01*, 2001.

[Del] G. Delzanno. *DMC : user guide.*

[DGG00] D. Dams, R. Gerth, and O. Grumberg. A heuristic for the automatic generation of ranking functions. In *Proceedings of Workshop on Advances in Verification, WAVe'00*, pages 1–8, 2000.

[Dil89] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 197–212, Berlin, June 1989. Springer.

[DKRT97] P. R. D'Argenio, J.-P. Katoen, T. C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time ! *Lecture Notes in Computer Science*, 1217 :416–432, 1997.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos. In *Workshop Hybrid Systems III : Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer-Verlag, oct 1996.

[DP] G. Delzanno and A. Podelski. Widen, narrow and relax. Working Draft.

[DP99] G. Delzanno and A. Podelski. Model checking in clp. In *Proc. the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS'99*, volume 1579, pages 223–239, 1999.

[DP00] G. Delzanno and A. Podelski. Constraint-based deductive model-checking. *Journal Software Tools for Technology Transfer*, 2000.

[DT98] C. Daws and S. Tripakis. Model-checking of real-time reachability properties using abstractions. In *Lecture Notes in Computer Science*, volume 1384, pages 313–329, 1998.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Hanbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science, 1990.

[FGK+96] J.C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. Cadp (cæsar / aldebaran development package) : A protocol validation and verification toolbox. In *Proc. of the 8th Computer Aided Verification, CAV'96*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440, 1996.

[FIS00] A. Finkel, S.P. Iyer, and G. Sutre. Well-abstracted transition systems : Application to fifo automata. Research Report LSV-00-6, ENS de Cachan, jun 2000.

[FJJM92] J-C. Fernandez, C. Jard, T. Jeron, and L. Mounier. "on-the-fly" verification of finite transition systems. In *Formal methods in system design*, 1992.

[Flo67] R.W. Floyd. Assigning meanings to programs. In *Proceedings Symposium on Applied Mathematics*, pages 19–32, 1967.

[FM96] A. Finkel and O. Marcé. Verification of infinite regular communicating automate. Technical report, LIFAC, Ecole Normale Supérieure de Cachan, France, 1996.

[FR96] L. Fribourg and J. Richardson. Symbolic verification with gap-order constraints. In *Proc. 6th International Workshop on Logic Program Synthesis and Transformation, LOPSTR'96*, Stockholm, Sweden, Aug. 1996.

[GL93] S. Graf and C. Loiseaux. A tool for symbolic program verification and abstraction. In *Computer Aided Verification CAV'93*, volume 697 of *Lecture Note in Computer Science*, pages 71–84, 1993.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proc. 2nd International Computer Aided Verification Conference*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185. Springer-Verlag, 1990.

[GS92] S. M. German and A. P. Sistla. Reasoning about systems with many processes. In *Journal of the ACM*, pages 675–735, 1992.

[GS97] S. Graf and H. Saidi. Construction of abstract state graphs with pvs. In *Conference on Computer Aided Verification CAV'97*, LNCS 1254, Springer Verlag, 1997.

[GW94] P. Godefroid and P. Wolper. A partial approach to model-checking. *Information and Computation*, 110 :305–326, 1994.

[Hab98] P. Habermehl. *Sur la Vérification de Systèmes Infinis*. PhD thesis, Université Joseph Fourier de Grenoble, 1998.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Proc. 3rd Int. Conf. Computer Aided Verification (CAV'93)*, volume 697 of *Lecture Notes in Computer Science*, pages 333–346, 1993.

[Hea99] A.C. Hearn. REDUCE — *User's and Contributed Packages Manual*, Feb. 1999.

[HH95] T. A. Henzinger and P-H. Ho. HYTECH : The cornell hybrid technology tool. In *Hybrid Systems II*, number 999 in Lecture Notes in Computer Science, pages 265–294, 1995.

[HHWT95] T. A. Henzinger, P-H. Ho, and H. Wong-Toi. HYTECH : The next generation. In *Proceedings of the 16th Annual IEEE Real-Time Systems Symposium, RTSS'95*, pages 56–65, 1995.

[HKPM97] G. Huet, G. Kahn, and C. Paulin-Mohring. The coq proof assistant - a tutorial, version 6.1. Technical report, INRIA, Aug 1997.

[HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111 :193–244, 1994.

[Hoa69] C.A.R Hoare. An axiomatic basis of computer programming. *Communications of the ACM*, 12, 1969.

[HRP94] N. Halbwachs, P. Raymond, and Y.E. Proy. Verification of linear hybrid systems by means of convex approximation. In *International Symposium on Static Analysis, SAS'94*, volume 864 of *Lecture Notes in Computer Science*, 1994.

[HRSV01] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. In *TACAS'01*, volume 2031 of *lncs*, pages 189–203, Italy, April 2001. Springer.

[HS96] Klaus Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. In *Formal Methods Europe FME '96*, number 1051 in Lecture Notes in Computer Science, pages 662–681. Springer-Verlag, March 1996.

[JJ89] T. Jeron and C. Jard. On-line model-checking for finite linear temporal logic. In *International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407, Grenoble, France, ju 1989.

[JN00] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *6th TACAS*, volume ? of *Lecture Notes in Computer Science*, page ?, 2000.

[KM97] J-P. Krimm and L. Mounier. Compositional state space generation from Lotos programs. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 239–258. Springer-Verlag, 1997.

[KMM+97] Y. Kersten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model-checking with rich assertional languages. In *Proceedings of 9th International Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 424–435, 1997.

[KP98] Y. Kesten and A. Pnueli. Modularization and abstraction : the keys to practical formal verification. In *23rd Int. Symp. Mathematical Foundations of Computer Science, MFCS-98*, volume 1450 of *Lecture Notes in Computer Science*, pages 54–71. Springer Verlag, 1998.

[Lab98] A. Labroue. Conditions de vivacité dans les automates temporisés. Technical Report LSV-98-7, ENS de Cachan, Laboratoire Spécification et Vérification, Sep. 1998.

[las] The liège automata-based symbolic handler (lash), available at http ://www.montefiore.ulg.ac.be/~boigelot/research/lash.

[Lon93] D.E. Long. *Model-checking, abstraction and compositional verification.* PhD thesis, Carnegie Mellon University, 1993.

[LP00a] K. Larsen and P. Pettersson. Uppaal2k. In *Bulletin of the European Association for Theoretical Computer Science*, volume 70, pages 40–44, 2000.

[LP00b] K. G. Larsen and P. Pettersson. Timed and hybrid systems in uppal2k. *MOVEP'2k*, 2000.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutschell. *Journal of Software Tools fot Technology Transfer*, 1(1-2) :134–152, 1997.

[LWYP98] K.G. Larsen, C. Weise, W. Yi, and J. Pearson. Clock difference diagrams. Technical report, Uppsala University Department of Computers Systems, 1998.

[Lyn96] *Distributed Algorithls.* Morgan Kaufmann Publishers, 1996.

[Mac93] K.L. MacMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[Mat96] Radu Mateescu. Formal description and analysis of a bounded retransmission protocol. Technical Report RR-2965, Inria, Institut National de Recherche en Informatique et en Automatique, 1996.

[May00] R. Mayr. Undecidable problems in unreliable computations. In *International Symposium on Latin American Theoretical Informatics, LATIN'2000*, volume 1776 of *Lecture Notes in Computer Science*, 2000.

[MB83] M. Measche and B. Berthomieu. Time petri-nets for analysing and verifying time dependent communication protocols. *Protocol Specification, Testing and Verification*, III, 1983.

[ML99] J. Moller and Jakob Lichtenberg. Difference decision diagram. Technical Report IT-TR-1999-023, Technical University of Denmark, feb 1999.

[MLAH99] J. Moller, Jakob Lichtenberg, H. Reif Andersen, and H. Hulgaard. On the symbolic verification of timed systems. Technical Report IT-TR-1999-024, Technical University of Denmark, feb 1999.

[MP91] Z. Manna and A. Pnueli. Completing the temporal picture. In *Theoratical Computer Science*, volume 83(1), pages 97–130, 1991.

[ORS92] S. Owre, J. Rushby, and N. Shankar. Pvs : a prototype verification system. In *11th Conf. on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752, 1992.

[Pac87] J.K. Pachl. Protocol designation and analysis based on a state transition model with channel expressions. *Protocol Specification, Testing, and Verification VII*, May 1987.

[Pel94] D. Peled. Combining partial order reductions with on-the-fly model-checking. In D. Dill, editor, *Computer Aided Verification CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 377–390. Springer-Verlag, 1994.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57, October 31–November 2 1977.

[Pnu85] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and models for concurrent systems*, 1985.

[Pre29] M. Presburger. Über die volstängigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes Rendus du Premier Congrès des Mathématiques des pays slaves*, 1929.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parametrized verification. In *Proceedings of Computer Aided Verification, CAV'00*, volume 1855 of *Lecture Notes in Computer Science*, pages 328–343, 2000.

[QS82] J.P. Queille and J. Sifakis. A temporal logic to deal with fairness in transition systems. In *23rd Annual symposium Foundations of Computer Science*, pages 217–225, 1982.

[STA98] R. Spelberg, H. Toetenel, and M. Ammerlaan. Partition refinement in real-time model-checking. In *Formal Techniques in Real-Time and Fault-Tolerance Systems*, volume 1486 of *Lecture Note Computer Sciences.* ?, 1998.

[Sut00] G. Sutre. *Abstraction et accélération de systèmes infinis.* PhD thesis, Ecole Normale Supérieure de Cachan, France, 2000.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems (extended abstract). In *13th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *lncs*, pages 347–359, Grenoble, France, February 1996. Springer.

[Tar51] Tarski. *A decision method for elementary algebra and geometry.* Universitu of California Press, second edition, 1951.

[Tea96] The Omega Team. *The Omega Library*, Nov. 1996.

[Tri98] S. Tripakis. *The Formal Analysis of Timed Systems in Practice.* PhD thesis, Université J. Fourier, Grenoble, 1998.

[Val89] A. Valmari. State space generation with induction. In *Proc. Scandinavian Conference on Artificial Intelligence*, pages 99–115, June 1989.

[VW86] M.Y. Vardi and P. Wolper. An Automata-Theroric Approach to Automatic Program Verification. In *Proc. 1st IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.

[Wei93] V. Weispfenning. A new approach to quantifier elimination for real algebra. Technical Report MIP-9305, Universität Passau, Germany, jul 1993.

[Win90] G. Winskel. Compositional checking of validity on finite state processes. In *Workshop on theories of communication*, volume 458, 1990.

[YKTB01] T. Yavuz-Kahveci, M. Tuncer, and T. Bultan. A library for composite symbolic representations. In *Proc. 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'01*, volume 2031 of *Lecture Notes in Computer Science*, pages 52–66, April 2001.

[Yov93] S. Yovine. *Méthodes et Outils pour la Vérification Symbolique de Systèmes Temporisés*. PhD thesis, Institut National Polytechnique de Grenoble, 1993.

[Yov97] S. Yovine. Kronos : a verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1/2), October 1997.

[Yov98] S. Yovine. Model checking timed automata. *Lecture Notes in Computer Science*, 1494 :114–153, 1998.

Annexe A

Preuves du chapitre 4

A.1 Preuve du lemme 2, page 62

Si $\tilde{M} = Nc$ alors $\tilde{M} \otimes \tilde{M} = \{\tilde{M}\}$, nous avons alors $[\![\tilde{M} \otimes \tilde{M}]\!] = [\![\tilde{M}]\!]$.

Supposons maintenant que $\tilde{M} \neq Nc$. Soit $\tilde{M} = (M, \phi)$. Soit $\{\tilde{M}'_1, \cdots, \tilde{M}'_n\} = \tilde{M} \otimes \tilde{M}$ tel que $\tilde{M}'_i = (M_i, \phi_i)$ pour tout i appartenant à $[1, n]$. D'après la définition de $\otimes$ nous avons

$$\forall i \in [1,n]\ \forall\ x, y \in \mathcal{X}\ \tilde{M}'_{ixy} = \bigoplus_{z \in \mathcal{X}} (M_{xz}, \phi) \otimes (M_{zy}, \phi)$$

- $\subseteq$ Puisque $\tilde{M} \in \mathcal{M}$, pour tout compteur $x \in \mathcal{X}$ nous avons $M_{xx} = (\leq, 0)$. Nous avons alors $(M_{xx}, \phi) \otimes (M_{xy}, \phi) = \{(M_{xy}, \phi)\}$. Donc pour chaque matrice i résultat nous avons $M'_{ixy} \sqsubseteq M_{xy}$ et par conséquent $\tilde{M}'_i \sqsubseteq \tilde{M}$. Ce qui donne $[\![\tilde{M}'_i]\!] \subseteq [\![\tilde{M}]\!]$ et enfin $\cup [\![\tilde{M}'_i]\!] \subseteq [\![\tilde{M}]\!]$ pour tout $i \in [1, n]$.
- $\supseteq$ Soit $\overrightarrow{\mathrm{d}} \in [\![\tilde{M}]\!]$. Par définition des matrices la formule $(x - y \prec M_{xy}\ \wedge\ \phi)[\mathcal{X}/\overrightarrow{\mathrm{d}}]$ est valide. Pour tout compteur $z \in \mathcal{X}$ les formules suivantes sont elles aussi valides :
  $\phi' = \quad (x - z \prec M_{xz}\ \wedge\ \phi)[\mathcal{X}/\overrightarrow{\mathrm{d}}]$
  $\phi'' = \quad (z - y \prec M_{zy}\ \wedge\ \phi)[\mathcal{X}/\overrightarrow{\mathrm{d}}]$
  Donc pour tout z la formule suivante est valide : $x - y \prec (M_{xz}, \phi) \otimes (M_{zy}, \phi)$. La formule étant vraie pour tout z, nous pouvons aussi dire que la formule $x - y \prec \oplus_{z \in \mathcal{X}} (M_{xz}, \phi) \otimes (M_{zy}, \phi)$ est valide. Par conséquent, il existe une matrice résultat $\tilde{M}'_i$ de $\tilde{M} \otimes \tilde{M}$ telle que la formule $(x - y \prec M'_{ixy}\ \wedge\ \phi'_i)[\mathcal{X}/\overrightarrow{\mathrm{d}}]$ est elle même valide. Donc $\overrightarrow{\mathrm{d}} \in [\![\tilde{M}_i]\!]$. Puisque nous avons fait le raisonnement pour n'importe quelle valuation, nous pouvons conclure que $[\![\tilde{M}]\!] \subseteq [\![\tilde{M} \otimes \tilde{M}]\!]$.

A.2 Preuve du lemme 4, page 63

La preuve est une preuve par contradiction (cf. [Yov93])
Nous supposons le contraire et nous trouvons une valuation $\overrightarrow{\mathrm{d}}$ pour laquelle il existe une matrice canonique $\tilde{M}^j$ ($j \in [1, n]$) telle que $\overrightarrow{\mathrm{d}} \in [\![\tilde{M}^j]\!]$ et $\overrightarrow{\mathrm{d}} \notin [\![\tilde{M}']\!]$. (ce qui contredit $[\![\tilde{M}]\!] = [\![\tilde{M}']\!]$ car $[\![cf(\tilde{M})]\!] = [\![\tilde{M}]\!]$ par le lemme 3).

(1) Supposons le contraire : $\exists j \in [1, n].\tilde{M}^j \not\sqsubseteq \tilde{M}'$.

Il existe alors 2 compteurs $x, y \in \mathcal{X}$ tels que $(M'_{xy}, \phi') \sqsubset (M^j_{xy}, \phi^j)$.

D'après la définition des matrices contraintes, nous avons $M_{zz} = M'_{zz} = (\leq, 0)$ pour tout $z \in \mathcal{X}$. Les compteurs x et y sont donc des compteurs différents.

Supposons que $M^j_{xy} = (\prec^j_{xy}, t^j_{xy})$, $M^j_{yx} = (\prec^j_{yx}, t^j_{yx})$ et $M'_{xy} = (\prec'_{xy}, t'_{xy})$.

Nous pouvons exprimer la relation $(M'_{xy}, \phi') \sqsubset (M^j_{xy}, \phi^j)$ de manière équivalente en posant que la formule suivante est valide :

$$\forall p \in \mathcal{P}\ (t'_{xy} < t^j_{xy}\ \wedge\ \phi^j\ \wedge\ \phi')\ \vee\ (t'_{xy} = t^j_{xy}\ \wedge\ \prec'_{xy} < \prec^j_{xy}\ \wedge \phi^j\ \wedge\ \phi') \tag{A.1}$$

Posons $t = [max(t'_{xy}, -t^j_{yx}) + t^j_{xy}]/2$.

Or, nous avons $((\leq, 0), vrai) \sqsubseteq ((\prec^j_{xy}, t^j_{xy}), \phi^j) \oplus ((\prec^j_{yx}, t^j_{yx}), \phi^j)$, cette relation est vraie si la formule A.2 est valide :

$$\forall p \in \mathcal{P}\ (-t^j_{yx} < t^j_{xy}\ \wedge\ \phi^j)\ \vee\ (t^j_{xy} = -t^j_{yx}\ \wedge\ \phi_j \wedge\ \prec^j_{xy} = \prec^j_{yx} = \leq) \tag{A.2}$$

Des deux formules A.1 et A.2, nous devons alors traiter les 4 cas suivants :

- Nous avons la formule $\forall p \in \mathcal{P}\ (t'_{xy} < t^j_{xy}\ \wedge\ \phi^j\ \wedge\ \phi'\ \wedge\ -t^j_{yx} < t^j_{xy})$ qui est valide. Par conséquent, nous avons

$$\forall p \in \mathcal{P}\ (t < t^j_{xy}\ \wedge\ max(t'_{xy}, -t^j_{yx}) < t)$$

- Nous avons la formule $\forall p \in \mathcal{P}\ (t'_{xy} < t^j_{xy}\ \wedge\ \phi^j\ \wedge\ \phi'\ \wedge\ -t^j_{yx} = t^j_{xy}\ \wedge\ \prec^j_{xy} = \prec^j_{yx} = \leq)$ qui est valide. Par conséquent, nous avons

$$\forall p \in \mathcal{P}\ (t = t^j_{xy}\ \wedge\ t = -t^j_{yx}\ \wedge\ t < t'_{xy})$$

- Nous avons la formule $\forall p \in \mathcal{P}\ (t'_{xy} = t^j_{xy}\ \wedge\ \phi^j\ \wedge\ \phi' \wedge\ \prec'_{xy} < \prec^j_{xy}\ \wedge\ -t^j_{yx} < t^j_{xy})$ qui est valide. Par conséquent, nous avons

$$\forall p \in \mathcal{P}\ (t = t^j_{xy}\ \wedge\ t = t'_{xy}\ \wedge\ t > -t^j_{yx} \wedge\ \prec'_{xy} = <))$$

- Nous avons la formule $\forall p \in \mathcal{P}\ (t'_{xy} = t^j_{xy}\ \wedge\ \phi^j\ \wedge\ \phi' \wedge\ \prec'_{xy} < \prec^j_{xy}\ \wedge\ -t^j_{yx} = t^j_{xy} \wedge\ \prec^j_{xy} = \prec^j_{yx} = \leq)$ qui est valide. Par conséquent, nous avons

$$\forall p \in \mathcal{P}\ (t = t^j_{xy}\ \wedge\ t = t'_{xy}\ \wedge\ t = -t^j_{yx})$$

Dans ces 4 cas, nous pouvons dire que la formule ci-dessous est valide :

$$\psi = \forall p \in \mathcal{P}\ (\phi^j\ \wedge\ \phi'\ \wedge\ -t \prec M^j_{xy}\ \wedge\ t \prec M^j_{yx}\ \wedge\ t \not\prec M'_{xy})$$

(2) L'expression t peut être utilisée pour construire une valuation $\overrightarrow{\mathrm{d}}$ telle que $\overrightarrow{\mathrm{d}} \in [\![\tilde{M}^j]\!]$ mais $\overrightarrow{\mathrm{d}} \notin [\![\tilde{M}']\!]$. La valuation $\overrightarrow{\mathrm{d}}$ est composée des valuations des compteurs d_c et de celles des paramètres d_p. La valuation $\overrightarrow{\mathrm{d}}$ est construite de la manière suivante.

Tout d'abord, nous instancions les valuations des paramètres. Cette valuation doit satisfaire la contrainte exprimée par ψ, on a alors $\overrightarrow{\mathrm{d}}_p \in [\![\psi]\!]$.

Construisons maintenant la partie concernant les compteurs. Nous donnons au compteur x la valeur de l'expression t (sous la valuation $\overrightarrow{\mathrm{d}}_p$) : $\overrightarrow{\mathrm{d}}(x) = (t)[\overrightarrow{\mathrm{d}}_p/\mathcal{P}]$. Pour les compteurs y et c_0, nous posons $\overrightarrow{\mathrm{d}}(y) = \overrightarrow{\mathrm{d}}(c_0) = 0$.

Supposons maintenant que la valuation $\overrightarrow{\mathrm{d}}$ soit définie pour un sous-ensemble des compteurs $\mathcal{X}^1 \subset \mathcal{X}$ (on aura alors $\mathcal{X}^1 = \mathcal{X}^1 \cup \mathcal{P}$) telle que pour tout $z, w \in \mathcal{X}$, la formule $(z - w \prec M^j_{zw})[\overrightarrow{\mathrm{d}}/\mathcal{X}^1]$.

Soit $r \in \mathcal{X}^1$. Nous devons construire la valuation de r de telle sorte que les formules
- $(r - w \prec M^j_{rw})[\overrightarrow{\mathrm{d}}/\mathcal{X}]$ et
- $(z - r \prec M^j_{zr})[\overrightarrow{\mathrm{d}}/\mathcal{X}]$

soient valides.

On trouvera une telle valeur si et seulement si $(((\leq, z - w), \phi^j) \sqsubseteq M^j_{zr} \otimes M^j_{rw})[\overrightarrow{\mathrm{d}}/\mathcal{X}]$ est valide. Or, cette inéquation est effectivement valide puisque $(z - w \prec M^j_{zw})[\overrightarrow{\mathrm{d}}/\mathcal{X}^1]$ l'est, et d'après la remarque 4.4 $M^j_{zw} \sqsubseteq M^j_{zr} \otimes M^j_{rw}$.

Par conséquent, la valuation $\overrightarrow{\mathrm{d}}$ ainsi construite satisfait $\overrightarrow{\mathrm{d}} \in [\![\tilde{M}^j]\!]$ mais $\overrightarrow{\mathrm{d}} \notin [\![\tilde{M}']\!]$. Nous aboutissons donc à une contradiction. Nous avons donc bien $\tilde{M}^j \sqsubseteq \tilde{M}'$.

A.3 Preuve de la propriété 6, page 65

Les deux premiers points sont immédiats.

3. pour tout $i, j \in [0, n_c]$
   $A_3[i,j] = \Sigma_{k=0}^{n_c} A_1[i,k] * A_2[k,j]$
   $= A_2[i',j]$ où $A_1[i,i'] = 1$
   Il existe un seul i' tel que $A_1[i,i'] = 1$ par définition de Mat^A. De la même manière pour une ligne donnée dans la matrice A_2 il n'existe qu'un seul j' tel que $A_2[i,j'] = 1$. Par conséquent $A_3[i,j] = 1$ si $A_1[i,i'] = 1$ et $A_2[i',j] = 1$ sinon $A_3[i,j] = 0$.
   A_3 est bien une matrice de Mat^A.
4. Soient $B_1, B_2 \in Mat^B$, soit $B_3 = B_1 + B_2$. Pour tout $i, j \in [0, n_c]$,
   - $B_3[i,j] \in AT(\mathcal{P})$.
   - $B_3[i,j] = B_1[i,j] + B_2[i,j] = -B_1[j,i] - B_2[j,i] = -B_3[j,i]$
   - pour tout $k \in [0, n_c]$, $B_3[i,k] + B_3[k,j] = B_1[i,k] + B_2[i,k] + B_1[k,j] + B_2[k,j] = B_1[i,j] + B_2[i,j] = B_3[i,j]$
5. Soient $A_1 \in Mat^A$ et $B_1 \in Mat^2$ telles que $B_2 = A_1 * B_2 * A_1^T$, nous avons alors pour tout $i, j \in [0, n_c]$:
   $B_2[i,j] = \Sigma_{l=0}^{n_c}(\Sigma_{k=0}^{n_c} A_1[i,k] * B_1[k,l]) * A_1^T[l,j]$
   $= \Sigma_{l=0}^{n_c}(\Sigma_{k=0}^{n_c} A_1[i,k] * B_1[k,l]) * A_1^T[j,l]$
   $= B_1[i',j']$ où $A[i,i'] = 1$ et $A[j,j'] = 1$
   De la même manière $B_2[j,i] = B_1[j',i'] = -B_1[i',j'] = -B_2[i,j]$.
   Enfin, pour tout $k \in [0, n_c]$,

$$\begin{aligned} B_2[i,k] + B_2[k,j] &= B_1[i',k'] + B_1[k',j'] \\ &= B_1[i',j'] \\ &= B_2[i,j] \end{aligned}$$

6. En effet, pour tout $i \in [0, n_c]$ nous avons $B[i,0] = -B[0,i]$ ce qui veut dire que $-B[0,i] = B[i,0] \leq x_i \leq B[i,0]$ où $x_i \in \mathcal{X}$. Ainsi pour chaque variable, la matrice B représente une valeur unique.

Annexe B

Analyse de “Fischer” sans contraintes

La configuration initiale est $\gamma_0 = ((repos^1, repos^2),\ 0 \leq h_1 \ \wedge\ 0 \leq h_2 \ \wedge\ h_1 = h_2,\ T > 0 \ \wedge\ D > 0$.

L'ensemble des configurations symboliques atteignables est présenté dans le tableau B.1 où $\phi_1 = T > 0 \ \wedge\ D > 0$, $\phi_2 = T + D > 0 \ \wedge\ D > 0 \ \wedge\ T > 0$, $\phi_3 = T < D \ \wedge\ D > 0 \ \wedge\ T > 0$, $\phi_4 = T > D \ \wedge\ D > 0 \ \wedge\ T > 0$, $\phi_5 = T = D \ \wedge\ D > 0 \ \wedge\ T > 0$. Les cases du tableau contenant plusieurs lignes doivent être lues comme une disjonction sur les valeurs des horloges et des paramètres.

Le graphe symbolique est donné sous le format Aldebaran [FGK+96]. Une transition est de la forme $(q_i, label, q_f)$ où q_i et q_f sont des numéros de configurations atteignables.

	état	id	$\tilde{M}$	contrainte
γ_0	$repos^1\ repos^2$	0	$h_1 \geq 0\ \wedge\ h_2 \geq 0\ \wedge\ h_1 = h_2$	ϕ_1
			$h_1 > T\ \wedge\ h_2 > T\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_2
			$h_1 > T\ \wedge\ h_2 > T\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_2
γ_1	$repos^1\ sessiond^2$	0	$0 \leq h_1,\ h_2 \leq D\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_2
γ_2	$repos^1\ attente^2$	0	$h_1 > T\ \wedge\ 0 \leq h_2\ \wedge\ T < h_1 - h_2 \leq D$	ϕ_2
			$T > h_1\ \wedge\ T > h_2\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_2
γ_3	$repos^1\ attente^2$	2	$0 \leq h_1\ \wedge\ 0 \leq h_2\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_2
γ_4	$repos^1\ sectionc^2$	0	$h_1 \geq 2T\ \wedge\ h_2 > T\ \wedge\ T < h_1 - h_2 \leq D$	ϕ_3
			$h_1 > T\ \wedge\ h_2 \geq 2T\ \wedge\ T < h_2 - h_1 \leq D$	ϕ_3
γ_5	$repos^1\ sectionc^2$	2	$h_1 > T\ \wedge\ h_2 > T\ \wedge\ 0 \leq h_1 - h_2 < D - T$	ϕ_4
			$0 \leq h_1\ \wedge\ T < h_2\ \wedge\ h_1 - h_2 < D - T$	ϕ_5
			$0 \leq h_1\ \wedge\ T < h_2\ \wedge\ h_1 - h_2 < -T$	ϕ_2
			$T < h_1\ \wedge\ T < h_2\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_2
γ_6	$sessiond^1\ repos^2$	0	$0 \leq h_1,\ h_2 \leq D\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_2
γ_7	$sessiond^1\ sessiond^2$	0	$0 \leq h_1,\ h_2 \leq D\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_2
			$0 \leq h_1,\ h_2 \leq D\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
γ_8	$sessiond^1\ attente^2$	0	$T < h_2\ \wedge\ 0 \leq h_1 \leq D\ \wedge\ h_2 - h_1 < T$	ϕ_2
			$0 \leq h_1 \leq D\ \wedge\ 0 \leq h_1\ \wedge\ h_1 - h_2 \leq 0$	ϕ_3
γ_9	$sessiond^1\ attente^2$	2	$0 \leq h_1,\ h_2 \leq D\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_2
γ_{10}	$sessiond^1\ sectionc^2$	0	$0 \leq h_1 \leq D\ \wedge\ T < h_2\ \wedge\ h_2 - h_1 < T$	ϕ_3
γ_{11}	$sessiond^1\ sectionc^2$	2	$T < h_1,\ h_2 \leq D\ \wedge\ 0 \leq h_2 - h_1 < D - T$	ϕ_3
γ_{12}	$attente^1\ repos^2$	0	$h_1 > T\ \wedge\ h_2 > T\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_2
			$0 \leq h_1\ \wedge\ T < h_2\ \wedge\ T < h_2 - h_1 \leq D$	ϕ_3
γ_{13}	$attente^1\ repos^2$	1	$0 \leq h_1\ \wedge\ 0 \leq h_2\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_2
γ_{14}	$attente^1\ sessiond^2$	0	$T < h_1\ \wedge\ 0 \leq h_2 \leq D\ \wedge\ h_2 - h_1 < -T$	ϕ_2
			$0 \leq h_1\ \wedge\ 0 \leq h_2 \leq D\ \wedge\ h_2 - h_1 \leq 0$	ϕ_3
γ_{15}	$attente^1\ sessiond^2$	1	$0 \leq h_1,\ h_2 \leq D\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_1
γ_{16}	$attente^1\ attente^2$	2	$0 \leq h_1\ \wedge\ 0 \leq h_2\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_1
γ_{17}	$attente^1\ attente^2$	1	$0 \leq h_1\ \wedge\ 0 \leq h_2\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_2
γ_{18}	$attente^1\ sectionc^2$	2	$h_1 > T\ \wedge\ h_2 > T\ \wedge\ 0 \leq h_1 - h_2 \leq D$	ϕ_2
γ_{19}	$attente^1\ sectionc^2$	1	$0 \leq h_1\ \wedge\ h_2 > T\ \wedge\ T < h_2 - h_1 \leq D$	ϕ_3
γ_{20}	$sectionc^1\ repos^2$	0	$h_1 \geq 2T\ \wedge\ T < h_2\ \wedge\ T < h_1 - h_2 \leq D$	ϕ_3
			$h_1 > T\ \wedge\ h_2 \geq 2T\ \wedge\ T < h_2 - h_1 \leq D$	ϕ_3
γ_{21}	$sectionc^1\ repos^2$	1	$h_1 \geq T\ \wedge\ 0 \leq h_2\ \wedge\ T < h_1 - h_2$	ϕ_2
			$h_1 > T\ \wedge\ h_2 > T\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_2
γ_{22}	$sectionc^1\ sessiond^2$	0	$T < h_1\ \wedge\ 0 \leq h_2 \leq D\ \wedge\ h_2 - h_1 < -T$	ϕ_3
γ_{23}	$sectionc^1\ sessiond^2$	1	$T < h_1,\ h_2 \leq D\ \wedge\ 0 \leq h_2 - h_1 \leq D - T$	ϕ_3
γ_{24}	$sectionc^1\ attente^2$	2	$T < h_1\ \wedge\ 0 \leq h_2\ \wedge\ T < h_1 - h_2 \leq D$	ϕ_3
γ_{25}	$sectionc^1\ attente^2$	1	$T < h_1\ \wedge\ T < h_2\ \wedge\ 0 \leq h_2 - h_1 \leq D$	ϕ_2
γ_{26}	$sectionc^1\ sectionc^2$	2	$2T \leq h_1\ \wedge\ T < h_2\ \wedge\ T < h_1 - h_2 \leq D$	ϕ_3
γ_{27}	$sectionc^1\ sectionc^2$	1	$T < h_1\ \wedge\ 2T \leq h_2\ \wedge\ T < h_2 - h_1 \leq D$	ϕ_3

FIG. B.1 – Ensemble des configurations atteignables du "Fischer"

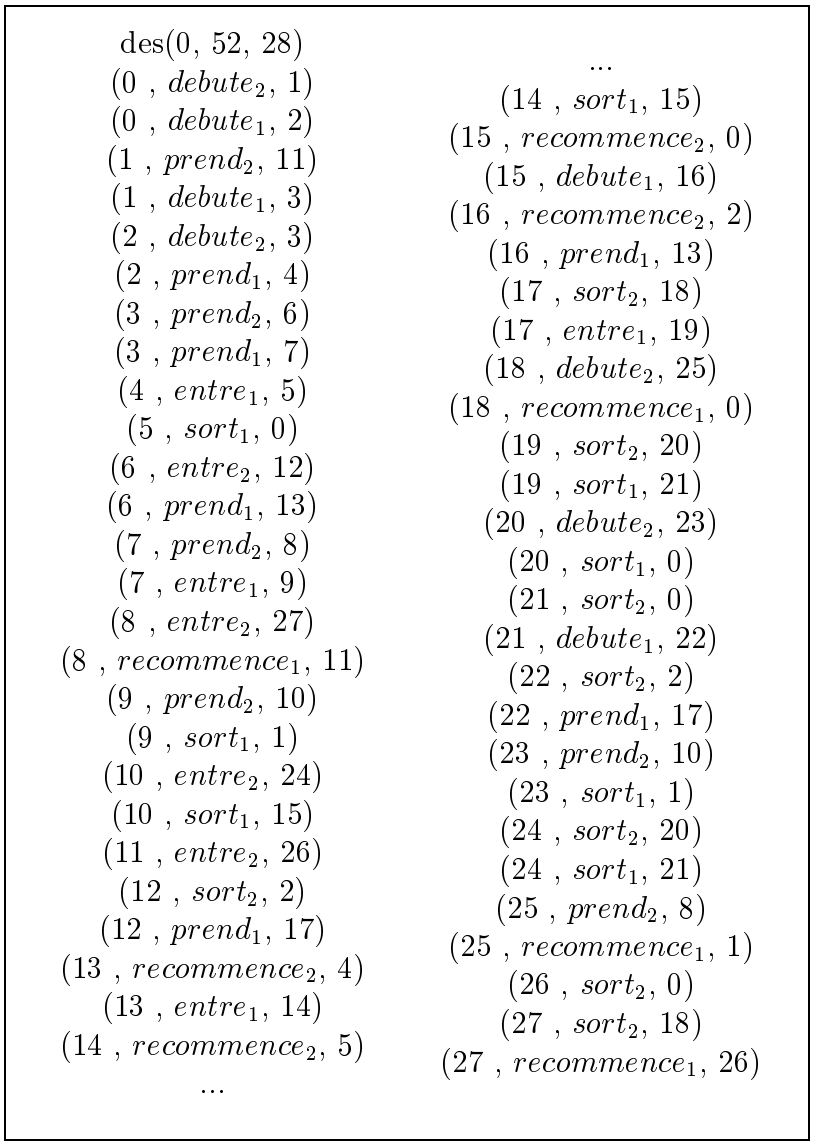


FIG. B.2 – Graphe symbolique de "Fischer" sans contraintes

Annexe C

TREX un outil d'analyse

C.1 Options de TREX

Les différentes options d'entrées pour notre outil.

-c *filename.cnd*

Utilise la contrainte spécifiée dans le fichier *filename.cnd* comme contrainte initiale sur les paramètres. La contrainte CND doit suivre la grammaire suivante :

$$\begin{array}{rcl} CND & ::= & F\ '\$' \\ F & ::= & T\ op\ T \mid not\ F \mid F\ and\ F \mid F\ or\ F \\ T & ::= & Constant \mid Variable \mid T * T \mid T + T \\ op & ::= & = \mid <= \mid < \mid >= \mid > \end{array}$$

où $Constant$ est l'ensemble des entiers et $Variable$ est un des paramètres.

-cf | -cfinit *filename.cf*

Utilise l'état de contrôle et l'ensemble des valeurs des variables donnés dans le fichier *filename.cf* comme configuration initiale pour l'analyse d'atteignabilité.

-cp *nombre*

Utilise *nombre* comme profondeur pour la recherche de boucles dans les algorithmes d'accélération. Par défaut la valeur de la profondeur est égale à 1.

-e | -elim

Utilise la librairie FOAF à la place de Reduce pour l'élimination des quantificateurs sur les réels. Cette option n'est pas valide par défaut.

-env

Réécrit toutes les opérations de la transition dans l'étiquette de cette transition. Cette option n'est pas valide par défaut.

-h | -help

Ecrit à l'écran un résumé de ces options.

-i | -init *filename.if*

Utilise le système décrit dans le fichier *filename.if* comme modèle initial pour l'analyse.

-ne | -nelim

Permet la substitution des variables d'itérations lorsqu'elles sont définies par une égalité dans l'ensemble des contraintes.

-pre
Utilise l'algorithme de l'analyse en arrière. Par défaut, c'est l'analyse en avant qui est utilisée.

-p | -prop *filename.if*
Utilise le système décrit dans le fichier *filename.if* comme observateur représentant une propriété de sûreté à vérifier. La vérification se fait alors "à la volée".

-red *nombre*
Utilise *nombre* comme nombre maximum d'appels à l'outil implantant les procédures de décisions sur les réels, Reduce. Une valeur usuelle est 10000. Cette option n'est pas valide par défaut.

-r | -res *filename.scf*
Une fois l'analyse terminée, écrit l'ensemble des configurations atteignables dans le fichier *filename.scf* selon le format décrit après les options.

-sg *filename.aut*
Ecrit le graphe symbolique fini représentant l'exécution du modèle initial sous le format Aldebaran dans le fichier *filename.aut*. Cette option n'est pas valide par défaut.

-s | -set
Considère la représentation des compteurs comme un ensemble. Cette option n'est pas valide par défaut.

-simpl
Appelle une procédure de simplification des formules arithmétiques après chaque opération sur ces formules. Cette option augmente considérablement le temps d'exécution mais peut réduire la place mémoire utilisée. Cette option n'est pas valide par défaut.

-t | -trace *filename.tr*
Ecrit la trace de l'exécution de l'analyse dans le fichier *filename.tr*.

Ainsi, TReX fournit l'ensemble des configurations atteignables du système dans le fichier *filename.scf*. Ce fichier contient au plus 3 parties distinctes : l'ensemble des configurations atteignables, la correspondance entre les états de contrôle du système et les états de contrôle du graphe symbolique (avec l'option -sg filename.aut) et le temps d'exécution. Nous donnons ci-après un exemple d'un fichier résultat pour un système complexe contenant des compteurs, des horloges, des files d'attente, des booléens et des paramètres :

```
========== Reachable Symbolic Configurations ==========
( system control state, property control state,
valuation of boolean and enumerative variables,
clocks valuations,
counters valuations,
channels contents,
constraints conjunction of parameters  ->
clocks valuation,
counters valuations,
channels contents,
constraints conjunction of parameters
```

```
)
 [...]

========== Correspondence with Symbolic Graph  ==========
[ symbolic graph control state -- ( system control state,
                                    valuation of boolean and enumerative variables,
                                    property control state) ]
[ ...]

========== Time Statistics ==========

child : x.xx
self  : x.xx
---------------
total : x.xx
```

L'ensemble des configurations symboliques est ordonné selon les états de contrôle de l'automate étendu (ce peut être l'association d'un état de contrôle d'un système associé à l'état de contrôle d'un système de transitions représentant une propriété (option -p filename.if)). Pour chaque état de contrôle il peut y avoir plusieurs ensembles de valuations pour les variables et les paramètres, ils sont alors séparés par le symbole "->".

Pour les statistiques de temps, on peut lire le temps passé dans les process fils de TREX à la ligne *child* (ce sera notamment le temps passé dans Omega ou Reduce) et le temps utilisé par l'exécution de TREXà la ligne $self$.

C.2 Grammaire de TREX

La grammaire utilisée par TREX est un sous-ensemble de la grammaire de IF [BGMS98]. Nous avons enlevé les cas suivants :

- Pas de synchronisations (autre que par variables booléennes).
- Pas de signaux avec des valeurs.

Voici la grammaire que nous utilisons selon le format BNF :

comment : := /* ... */
int-constant : := [0-9]+
id : := [a-zA-Z][_a-zA-Z0-9]*
system-def : := **system** *system-id* ';'
signal *signal-id* ';' { *signal-id* ';'}
var *variable-def* {*variable-def* }
buffer *buffer-def* {*buffer-def*}
process-def {*process-def*}
predef-type-id : := **bool**
| **int**
| **real**
| **clock**
type-def : := *type-id* '=' *type-spec* ';'
type-spec : := **enum** {',' *item-id*}
variable-def : := *variable-id* [*initializer*] ' :' *type-id* ';'
initializer : := '(' *expression* ')'
buffer-def : := *buffer-id* ' :' **queue** ' :'
[**toenv lossy**] **of** *signal-id* {',' *Signal-id*}
process-def : := **process** *process-id* ';'
[**var** *variable-def* {*variable-def*}]
state *state-def* {*state-def*}
transitions *trans-def* {*trans-def*}
state-def : := *state-id* { ' :'**init, ' :'nostable}**
[**tpc** *expression* ';' **end**] ';'
transitions : := **from** *stat-id*
[**if** *expression*] [*input*] [**do** *body-action*]
to *state-id* ';'
input : := **input** *signal-id* **from** *buffer-id*
body-action : := *body-action* ',' *body-action*
| **output** *signal-id* **to** *buffer-id*
| *expression* ' :=' *expression*
| [**reset**] *expression*
expression : := *constant*
| *variable-id*
| *unary-op expression*
| *expression binary-op expression*
| '(' *expression* ')'
constant : := *int-constant*
| *item-id*
unary-op : := **not**
| **−** | +
binary-op : := * | / | %
| **−** | +
| > | < | <= | >= | = | <>
| **and** | **or**

C.3 Algorithme de canonisation symbolique

Dans l'algorithme de la canonisation que nous présentons ci-après (cf. figures C.1, C.2 et C.3) nous avons choisi les conventions suivantes :

- *taille* est la dimension de la matrice canonisée.
- Tous les éléments du tableau T sont des matrices contraintes donc $T[l] = \tilde{M} = (M, \phi)$. Nous notons par $T[l].\tilde{M}[i,j]$ la borne paramétrée contrainte $((\prec_{ij}, t_{ij}), \phi)$ associée au ijème élément de la matrice contrainte $\tilde{M}$. Nous notons par $T[l].M[i,j]$ la borne paramétrée $(\prec_{ij}, t_{ij})$ associée au ijème élément de la matrice M.

fonction Canonisation(Matrice contrainte $\tilde{M}$) : liste de matrices contraintes
lexique :
i, j, k, $nbcasesocc$: entier (0, 0, 0, 0, 1)
T : tableau de matrices contraintes
l, li, lj : entier
algo. :
$T[0] := \tilde{M}$
tant que ($(k < taille)$ **et puis** $(nbcasesocc \neq 0)$)
$i := 0$
tant que ($(i < taille)$ **et puis** $(nbcasesocc \neq 0)$)
$j := 0$
tant que ($(j < taille)$ **et puis** $(nbcasesocc \neq 0)$)
pour toutes les cases l **de** T **occupées :**
res := $min(T[l].\tilde{M}[i,j],\ T[l].\tilde{M}[i,k] + T[l].\tilde{M}[k,j])$
selon res
1. $t_{ij} < t_{ik} + t_{kj}$:
si $(ajouter\_contrainte(T[l],\ t_{ij} < t_{ik} + t_{kj}) = 0)$
alors $nbcasesocc - -$
2. $t_{ij} = t_{ik} + t_{kj} \wedge \prec_{ij} \leq \prec_{ik+kj}$:
si $(ajouter\_contrainte(T[l],\ t_{ij} = t_{ik} + t_{kj}) = 0)$
alors $nbcasesocc - -$
3. $t_{ij} = t_{ik} + t_{kj} \wedge \prec_{ij} \geq \prec_{ik+kj}$:
$T[l].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
si ($(ajouter\_contrainte(T[l],\ t_{ij} = t_{ik} + t_{kj}) = 0)$
ou alors
$((i = j)$ **et puis** $(T[l].\tilde{M}[i,j] < (0, vrai))))$
alors $nbcasesocc - -$
4. $t_{ij} > t_{ik} + t_{kj}$:
$T[l].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
si ($(ajouter\_contrainte(T[l],\ t_{ij} > t_{ik} + t_{kj}) = 0)$
ou alors
$((i = j)$ **et puis** $(T[l].\tilde{M}[i,j] < (0, vrai))))$
alors $nbcasesocc - -$
5. $t_{ij} \leq t_{ik} + t_{kj} \wedge \prec_{ij} \leq \prec_{ik+kj}$:
si $(ajouter\_contrainte(T[l],\ t_{ij} \leq t_{ik} + t_{kj}) = 0)$
alors $nbcasesocc - -$

[...]

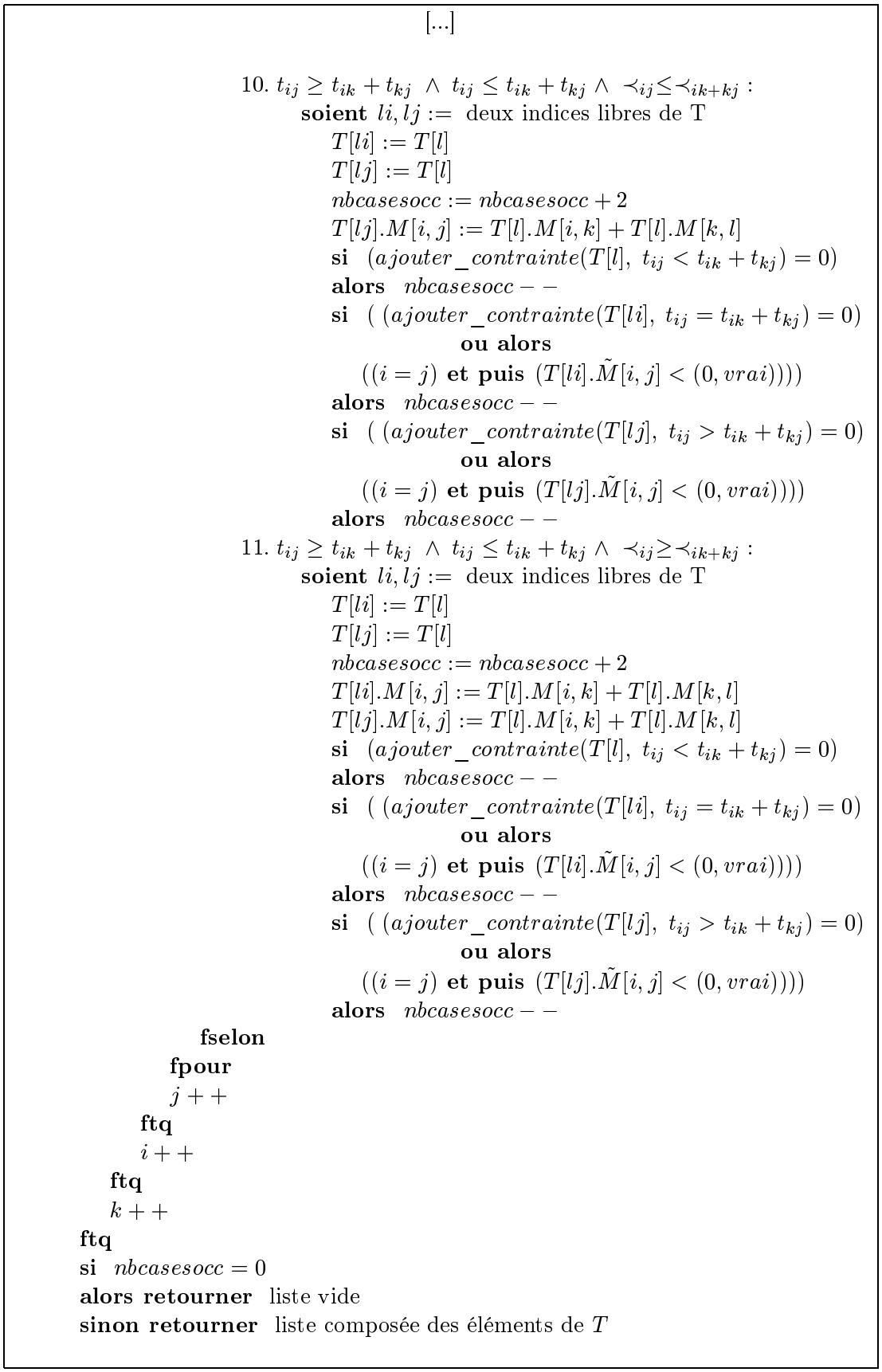
FIG. C.1 – Algorithme de canonisation de matrices de bornes paramétrées contraintes (début)

[...]

6. $t_{ij} \leq t_{ik} + t_{kj} \wedge \prec_{ij} \geq \prec_{ik+kj}$:
   soit $li :=$ indice libre de T **dans**
   $T[li] := T[l]$
   $nbcasesocc := nbcasesocc + 1$
   $T[li].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
   si $(ajouter\_contrainte(T[l],\ t_{ij} < t_{ik} + t_{kj}) = 0)$
   alors $nbcasesocc - -$
   si ($(ajouter\_contrainte(T[li],\ t_{ij} = t_{ik} + t_{kj}) = 0)$
   ou alors
   $((i = j)$ **et puis** $(T[li].\tilde{M}[i,j] < (0, vrai))))$
   alors $nbcasesocc - -$
7. $t_{ij} < t_{ik} + t_{kj} \ \wedge\ t_{ij} > t_{ik} + t_{kj}$:
   soit $li :=$ indice libre de T
   $T[li] := T[l]$
   $nbcasesocc := nbcasesocc + 1$
   $T[li].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
   si $(ajouter\_contrainte(T[l],\ t_{ij} < t_{ik} + t_{kj}) = 0)$
   alors $nbcasesocc - -$
   si ($(ajouter\_contrainte(T[li],\ t_{ij} > t_{ik} + t_{kj}) = 0)$
   ou alors
   $((i = j)$ **et puis** $(T[li].\tilde{M}[i,j] < (0, vrai))))$
   alors $nbcasesocc - -$
8. $t_{ij} \geq t_{ik} + t_{kj} \wedge \prec_{ij} \geq \prec_{ik+kj}$:
   $T[l].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
   si ($(ajouter\_contrainte(T[l],\ t_{ij} \geq t_{ik} + t_{kj}) = 0)$
   ou alors
   $((i = j)$ **et puis** $(T[l].\tilde{M}[i,j] < (0, vrai))))$
   alors $nbcasesocc - -$
9. $t_{ij} \geq t_{ik} + t_{kj} \wedge \prec_{ij} \leq \prec_{ik+kj}$:
   soit $li :=$ indice libre de T
   $T[li] := T[l]$
   $nbcasesocc := nbcasesocc + 1$
   $T[li].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
   si $(ajouter\_contrainte(T[l],\ t_{ij} = t_{ik} + t_{kj}) = 0)$
   alors $nbcasesocc - -$
   si ($(ajouter\_contrainte(T[li],\ t_{ij} > t_{ik} + t_{kj}) = 0)$
   ou alors
   $((i = j)$ **et puis** $(T[li].\tilde{M}[i,j] < (0, vrai))))$
   alors $nbcasesocc - -$

[...]

FIG. C.2 – Algorithme de canonisation de matrices de bornes paramétrées contraintes (milieu)



[...]

10. $t_{ij} \geq t_{ik} + t_{kj} \ \wedge \ t_{ij} \leq t_{ik} + t_{kj} \ \wedge \ \prec_{ij} \leq \prec_{ik+kj}$:
soient $li, lj :=$ deux indices libres de T
$T[li] := T[l]$
$T[lj] := T[l]$
$nbcasesocc := nbcasesocc + 2$
$T[lj].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
si $(ajouter\_contrainte(T[l],\ t_{ij} < t_{ik} + t_{kj}) = 0)$
alors $nbcasesocc - -$
si ($(ajouter\_contrainte(T[li],\ t_{ij} = t_{ik} + t_{kj}) = 0)$
ou alors
$((i = j)$ **et puis** $(T[li].\tilde{M}[i,j] < (0, vrai))))$
alors $nbcasesocc - -$
si ($(ajouter\_contrainte(T[lj],\ t_{ij} > t_{ik} + t_{kj}) = 0)$
ou alors
$((i = j)$ **et puis** $(T[lj].\tilde{M}[i,j] < (0, vrai))))$
alors $nbcasesocc - -$
11. $t_{ij} \geq t_{ik} + t_{kj} \ \wedge \ t_{ij} \leq t_{ik} + t_{kj} \ \wedge \ \prec_{ij} \geq \prec_{ik+kj}$:
soient $li, lj :=$ deux indices libres de T
$T[li] := T[l]$
$T[lj] := T[l]$
$nbcasesocc := nbcasesocc + 2$
$T[li].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
$T[lj].M[i,j] := T[l].M[i,k] + T[l].M[k,l]$
si $(ajouter\_contrainte(T[l],\ t_{ij} < t_{ik} + t_{kj}) = 0)$
alors $nbcasesocc - -$
si ($(ajouter\_contrainte(T[li],\ t_{ij} = t_{ik} + t_{kj}) = 0)$
ou alors
$((i = j)$ **et puis** $(T[li].\tilde{M}[i,j] < (0, vrai))))$
alors $nbcasesocc - -$
si ($(ajouter\_contrainte(T[lj],\ t_{ij} > t_{ik} + t_{kj}) = 0)$
ou alors
$((i = j)$ **et puis** $(T[lj].\tilde{M}[i,j] < (0, vrai))))$
alors $nbcasesocc - -$
fselon
fpour
$j++$
ftq
$i++$
ftq
$k++$
ftq
si $nbcasesocc = 0$
alors retourner liste vide
sinon retourner liste composée des éléments de T

FIG. C.3 – Algorithme de canonisation de matrices de bornes paramétrées contraintes (fin)

Résumé : Dans le cadre de la télécommunication, les entreprises développent des protocoles gérant le transfert de données entre machines. Ces protocoles fonctionnent sur le principe d'envoi de messages entre deux parties par l'intermédiaire de canaux non fiables. Pour s'assurer que tous les messages ont bien été reçus, les techniques employées consistent à réémettre les messages perdus et/ou à attendre un laps de temps déterminé avant de conclure à l'échec de la transmission. De plus, les systèmes sont souvent modélisés en fonction de paramètres. Nous avons travaillé sur un modèle mathématique permettant la vérification de spécifications (comportements attendus des systèmes) pour des protocoles manipulant à la fois des compteurs, des files d'attente ou des horloges, ainsi que des paramètres. Le but de l'analyse est de calculer l'ensemble des comportements possibles du système puis de vérifier qu'aucun d'eux ne viole une spécification attendue. Le problème ici est que cet ensemble est infini. En effet, un comportement est fonction des valeurs prises par les variables du système au cours de l'exécution et certaines sont définies sur un domaine infini. Il faut alors pouvoir représenter ces comportements de façon finie et aussi trouver des méthodes pour calculer en un temps fini un ensemble infini. Plus formellement, nous nous sommes placés dans le cadre de l'analyse automatique des systèmes (model-checking). La représentation choisie pour les modèles à compteurs et horloges paramétrés est une extension des matrices de bornes pour laquelle nous avons une méthode exacte d'accélération (calcul en un temps fini d'ensembles de comportements infinis). Du côté pratique, nous avons implanté ces méthodes dans un outil TReX qui est, à notre connaissance, le seul pouvant manipuler de manière exacte des compteurs, des horloges et des files d'attente. Nous avons pu vérifier des exemples conséquents tels que le protocole de retransmission bornée.

Mots-clés : Abstraction, Model-checking, Paramètres, Protocoles de communication, Systèmes à compteurs, Systèmes temps-réels, Vérification.

Abstract : Within the framework of telecommunication, the companies develop protocols performing the data transfer between machines. These protocols work on the principle of sending of messages between two parts via unreliable channels. To make sure that all the messages were received, the techniques employed consist in re-emitting the lost messages and/or awaiting a lapse of time determined before deciding of the failure from the transmission. Moreover, the systems are often modelled according to parameters. We worked on a mathematical model allowing the verification of specifications (awaited behaviors of the systems) for protocols handling at the same time counters, queues or clocks, as well as parameters. The goal of the analysis is to calculate the set of the behaviors possible for the system then to check that none them violates an awaited specification. The problem here is that this set is infinite. Indeed, a behavior is a function of the values taken by the variables of the system during the execution and some are defined on an infinite field. It is then necessary to be able to represent these behaviors in a finished way and to also find methods to calculate in a finished time an infinite set. More formally, we placed ourselves within the framework of the automatic analysis of the systems (model-checking). The selected representation for the models with counters and clocks parameterized is an extension of the bounded matrices for which we have an exact method of acceleration (calculation in a finished time of sets of infinite behaviors). Practical side, we established these methods in a tool TReX which is, to our knowledge, only being able to handle in an exact way of the counters, the clocks and the queues. We could check consequent examples such as the bounded retransmission protocol.

Keywords : Abstraction, Communication protocols, Counters systems, Model-checking, Parameters, Real-time systems, Verification.