RAPPORT DE DEA/TFE

Vérification de réseaux de Petri temporels étendus à l'aide de polyèdres

Morgan Magnin

Responsables : Olivier H. Roux et Didier Lime

Institut de Recherche en Communications et Cybernétique de Nantes
Ecole Centrale de Nantes

Septembre 2004

Table des matières

Introduction

Les architectures embarquées nécessitent généralement que calculs et vérification soient exécutés en temps réel. Par « temps réel », nous entendons le fait que le système soit capable de répondre à des sollicitations de l'environnement en des temps de réponse en relation avec la constante de temps dominante dans cet environnement. Les domaines concernés se caractérisant souvent par une haute criticité (nucléaire, aéronautique, etc.), il est indispensable de développer des outils de modélisation et de vérification formelle.

Les réseaux de Petri T-temporels [Mer74] et les automates hybrides [Hen96] constituent deux de ces outils. Les premiers sont particulièrement adaptés à la modélisation de systèmes temps réels tandis que les seconds sont appréciés pour les outils de vérification qui leur sont associés.

Seulement, les réseaux de Petri T-temporels ne permettent pas, dans leur formulation initiale, de prendre en compte l'ordonnancement des différentes entités logicielles de l'application réparties sur une architecture matérielle multi-processeurs. C'est ainsi qu'une extension des réseaux de Petri T-temporels, appelée réseaux de Petri temporels étendus à l'ordonnancement (SETPN), a été proposée pour tenir compte de préemptions liées à des priorités fixes [RD01]. Toutefois, les analyses d'accessibilité et de bornitude pour les SETPN sont des problèmes indécidables. Le calcul des classes d'états et la vérification des SETPN met en jeu des mécanismes identiques à ceux rencontrés lors de l'étude des automates hybrides.

Dans notre séminaire bibliographique, nous avons étudié en détails les mécanismes de vérification formelle des automates hybrides, ainsi que les méthodes permettant de calculer des approximations de l'espace d'accessibilité de tels automates. Durant la partie pratique de notre stage, nous nous sommes focalisés sur les SETPN. Pour ce modèle, nous avons proposé une méthode et un algorithme de calcul de l'ensemble des états accessibles (espace d'états). Cet ensemble est représenté par un graphe pour lequel à chaque noeud est associé un système d'inéquations linéaires sur les variables « de temps » (polyèdre). L'algorithme proposé a été implémenté en C à l'aide de New Polka [Jea02], une bibliothèque standard de manipulation des polyèdres.

Dans le chapitre 1 de ce rapport, nous présentons les réseaux de Petri temporels étendus à l'ordonnancement et les algorithmes propres au calcul de l'espace d'états. Le chapitre 2 propose un aperçu des différentes bibliothèques de manipulation des polyèdres et quelques détails sur l'implémentation, en C++, des algorithmes que nous avons développés. Enfin, le chapitre 3 fait le point sur les tests que nous avons menés et sur les conclusions que nous en avons tirées.

Chapitre 1

Réseaux de Petri Temporels étendus à l'Ordonnancement

Dans ce chapitre, nous présenterons d'abord le formalisme des réseaux de Petri temporels étendus à l'ordonnancement tel que l'ont introduit Olivier ROUX et Anne-Marie DÉPLANCHE [RD01]. Nous détaillerons ensuite une méthode de calcul exact de l'espace d'états. Puis nous reviendrons sur une méthode de surapproximation de l'espace d'états, méthode qui permet une représentation abstraite compacte des domaines de tir à l'aide de DBM (*Difference Bound Matrix*) [LR03a].

1.1 Introduction

ROUX et DÉPLANCHE ont proposé une extension des réseaux de Petri [RD01] qui prend en compte la manière dont sont réparties les tâches temps réel d'une application distribuée sur différents processeurs. Pour une politique d'ordonnancement à priorités fixes, le modèle SETPN introduit deux nouveaux attributs (γ et ω) associés à chaque place : le premier représente le processeur alloué, le second la priorité de la tâche modélisée. Etant donné un marquage M, une fonction Act permet de représenter le comportement de l'ordonnanceur de la manière suivante : si nous supposons que la place P modélise un comportement (ou un état) de la tâche T, $M(p) > 0$ signifie que la tâche T est prête et $Act(M(p)) > 0$ signifie que la tâche T est active.

Il n'est pas nécessaire de définir ces paramètres pour toutes les places d'un SETPN. En fait, quand une place ne représente pas une activité réelle du processeur (par exemple un registre ou un état mémoire), aucun processeur (γ) ni aucune priorité (ω) ne doit lui être attaché. Dans ce cas particulier, la sémantique est inchangée par rapport à celle d'un réseau de Petri temporel standard. Ceci est équivalent au fait d'associer à chaque place un processeur pour son usage exclusif et une priorité quelconque.

1.2 Définition et sémantique

1.2.1 Définition

Définition 1.1 (Réseau de Petri temporel étendu à l'ordonnancement). *Un* réseau de Petri temporel étendu à l'ordonnancement *(SETPN) est un n-tuple* $T = (P, T, pre, post, \alpha, \beta, M_0, Act)$ *tel que :*

- $P = \{p_1, p_2, \ldots, p_m\}$ *est un ensemble fini non-vide de* places.

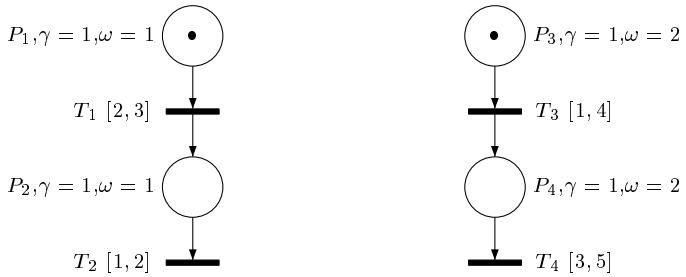

FIG. 1.1 – SETPN de deux tâches réparties sur un processeur

- $T = \{t_1, t_2, \ldots, t_n\}$ *est un ensemble fini non-vide de* transitions *($T \cap P = \emptyset$).*
- $pre \in (\mathbb{N}^P)^T$ *est la fonction d'incidence arrière.*
- $post \in (\mathbb{N}^P)^T$ *est la fonction d'incidence avant.*
- $M_0 \in \mathbb{N}^P$ *est le marquage initial du réseau.*
- $\alpha \in (\mathbb{Q}^+)^T$ *et* $\beta \in (\mathbb{Q}^+ \cup \{\infty\})^T$ *sont les fonctions donnant respectivement, pour chaque transition, sa date de tir au plus tôt et au plus tard ($\alpha \leq \beta$).*
- $Act \in (\mathbb{N}^P)^{\mathbb{N}^P}$ *est la fonction de marquage actif. $Act(M)$ est la projection de la stratégie d'ordonnancement sur le marquage M. Dans [RD01], $Act(M)$ est définie pour une politique d'ordonnancement à priorité fixe à partir de trois paramètres :*
  - $Proc\{\phi, proc_1, proc_2, \ldots proc_l\}$ *est un ensemble fini de processeurs (dont ϕ permet de spécifier qu'une place n'est assigné à aucun processeur effectif de l'architecture matérielle).*
  - $\omega \in \mathbb{N}^P$ *est la fonction d'assignement de priorité.*
  - $\gamma \in Proc^P$ *est la fonction d'allocation.*

  Soit $E_{ma} = \{ma_1, ma_2 \ldots, ma_s\}$ (où $\forall i, ma_i \leq M$) l'ensemble des marquages admissibles associés à un marquage M donné et qui ne prend pas en compte les priorités affectées aux places. Le marquage d'une place $p \in P$ est actif ssi :
  - *La place p sensibilise au moins une transition t ;*
  - *Aucune autre place associée au même processeur n'est marquée dans ma.*

  Le marquage actif *$Act(M)$ est donc l'élément de E_{ma} tel que, pour chacune de ses places marquées, il n'existe pas un autre marquage admissible tel qu'une place associée au même processeur et plus prioritaire y est marquée.*

Act constitue l'élément spécifique permettant d'étendre les réseaux de Petri temporels aux SETPN.

Exemple 1.1. *La figure 1.1 présente un SETPN standard pour lequel deux tâches sont réparties sur un même processeur. Les activités associées aux places P_3 et P_4 sont prioritaires devant celles des places P_1 et P_2.*

1.2.2 Sémantique

En s'inspirant des travaux présentés dans [LPY95], la sémantique d'un SETPN peut être définie en terme de « système de transition temporisé ». Ces systèmes constituent des systèmes de transition standard avec deux types de transitions : des transitions discrètes associées aux événements (tirs de transitions) et des transitions continues pour l'écoulement du temps.

Un *marquage* M du réseau est un élément de $\mathbb{N}^P$ tel que $\forall p \in P, M(p)$ est le nombre de jetons dans la place p.

Un *marquage actif* $Act(M)$ du réseau est un élément de $\mathbb{N}^P$ tel que $\forall p \in P, Act(M(p)) = M(p)$ ou $Act(M(p)) = 0$.

Une transition t est dite *sensibilisée* par le marquage M si $M \geq pre(t)$ (*i.e.* si le nombre de jetons du marquage M dans chaque place d'entrée de t est supérieur ou égal au poids de l'arc entre cette place et cette transition). On le note $t \in enabled(M)$.

Une transition t est dite *active* si elle est sensibilisée par le marquage actif $Act(M)$. On le note $t \in enabled(Act(M))$.

Une transition t_k est dite *nouvellement* sensibilisée par le tir de la transition t_i à partir du marquage M, et nous le notons $\uparrow enabled(t_k, M, t_i)$ si la transition est sensibilisée par le nouveau marquage $M - pre(t_i) + post(t_i)$ mais pas par $M - pre(t_i)$ où M est le marquage du réseau avant le tir de t_i. Formellement, cela s'écrit :

$$\uparrow enabled(t_k, M, t_i) = (pre(t_k) \leq M - pre(t_i) + post(t_i)) \wedge ((t_k = t_i) \vee (pre(t_k) > M - pre(t_i)))$$

Par extension, nous notons $\uparrow enabled(M, t_i)$ l'ensemble des transitions nouvellement sensibilisées par le tir de la transition t_i à partir du marquage M.

Définition 1.2. *La sémantique d'un réseau de Petri temporel étendu à l'ordonnancement $\mathcal{T}$ est définie comme un système de transition temporisé $\mathcal{S}_{\mathcal{T}} = (Q, q_0, \rightarrow)$ [LPY95] tel que :*

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$
- $q_0 = (M_0, \bar{0})$ *($\bar{0}$ est la valuation initiale)*
- $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ *est la relation de transition incluant une relation de transition continue et une relation de transition discrète.*
  - *La relation de transition continue est définie $\forall d \in \mathbb{R}^+$ par :*

$$(M, \nu) \xrightarrow{d} (M, \nu') \text{ ssi}$$
$$\begin{cases} \forall t_i \in enabled(M), \nu'(t_i) = \begin{cases} \nu(t_i) \text{ si } Act(M) < pre(t_i) \wedge M \geq pre(t_i) \\ \nu(t_i) + d \text{ sinon,} \end{cases} \\ \forall t_k \in T, M \geq pre(t_k) \Rightarrow \nu'(t_k) \leq \beta(t_k) \end{cases}$$

  - *La relation de transition discrète est définie $\forall t_i \in T$ par :*

$$(M, \nu) \xrightarrow{t_i} (M', \nu') \text{ iff ,}$$
$$\begin{cases} Act(M) \geq pre(t_i), \\ M' = M - pre(t_i) + post(t_i), \\ \alpha(t_i) \leq \nu(t_i) \leq \beta(t_i), \\ \forall t_k, \nu'(t_k) = \begin{cases} 0 \text{ si } \uparrow enabled(t_k, M, t_i), \\ \nu(t_k) \text{ sinon} \end{cases} \end{cases}$$

1.2.3 Accessibilité et bornitude

Dans [RD01], ROUX et DÉPLANCHE ont démontré le résultat suivant concernant l'accessibilité et la bornitude des SETPN.

Théorème 1.1. *Les problèmes d'accessibilité et de bornitude sont indécidables pour les SETPN.*

Démonstration 1.1. *Un réseau de Petri temporel est un cas particulier d'un SETPN. Par ailleurs,* MENASCHE *a montré que les réseaux de Petri et les réseaux de Petri avec arcs inhibiteurs peuvent être décrits comme des réseaux de Petri temporels [Men82], et, par conséquent, comme des SETPN. Or les propriétés d'accessibilité et de bornitude sont indécidables pour les réseaux de Petri avec des arcs inhibiteurs et donc pour les SETPN.*

Toutefois, comme pour les réseaux de Petri temporels [BD91], si le réseau de Petri sous-jacent est borné, alors le SETPN est borné.

1.3 Calcul du graphe des classes

1.3.1 Cas des réseaux de Petri temporels standards

L'analyse d'un réseau de Petri temporel passe par le calcul de l'espace d'états accessible. Toutefois, l'espace d'états d'un réseau de Petri temporel étant, de manière évidente, infini, BERTHOMIEU et DIAZ ont proposé, pour un réseau de Petri temporel borné, une méthode permettant de calculer l'espace d'états comme un ensemble fini de *classes d'états* [BD91]. Une classe d'états contient tous les états du réseau entre le tir de deux transitions successives.

Définition 1.3. *Une* classe d'états *C d'un réseau de Petri temporel est un couple (M, D) où M est un marquage du réseau et D un ensemble d'inéquations appelé* domaine de tir.

Les inéquations de D sont de deux types :

$$\begin{cases} \alpha_i \leq x_i \leq \beta_i \ (\forall i \text{ tel que } t_i \text{ est sensibilisée}), \\ -\gamma_{kj} \leq x_j - x_k \leq \gamma_{jk}, \ \forall j,k \text{ tels que } j \neq k \text{ et } (t_j, t_k) \in enabled(M)^2 \end{cases}$$

x_i représente le temps de tir de la transition sensibilisée t_i relativement à l'instant où l'on est entré dans la classe.

Comme nous le verrons dans le chapitre 2, tout polyèdre, et donc tout domaine D défini par de telles inéquations, peut être mis sous forme canonique.

Nous avons vu que les inéquations définissant le domaine d'une classe d'un réseau de Petri temporel sont de la forme $x_i - x_j \leq \gamma_{ij}$ et $\alpha_k \leq x_k \leq \beta_k$ avec $i, j \in [\![1, n]\!]$ et $k \in [\![1, m]\!]$. Notons x_0 une variable telle que $x_0 = 0$. Nous pouvons écrire toutes les inéquations précédentes sous la forme $x_i - x_j \leq \gamma'_{i,j}$ avec $i, j \in [\![0, n]\!]$. Le système peut donc être représenté par la matrice $\mathcal{D} \in \mathcal{M}_{n+1}(\mathbb{R})$ telle que si d_{ij} est l'élément de $\mathcal{D}$ situé sur la i-ème ligne et la j-ème colonne, $d_{ij} = \gamma'_{ij}$. $\mathcal{D}$ est une *matrice de bornes de différences* ou *difference bound matrix* (DBM) en anglais.

Dans le cas général, la mise sous forme canonique d'un polyèdre faisant intervenir n variables a une complexité au pire cas exponentielle en n [AFP02]. Par contre, pour les DBM, l'inclusion, l'intersection et l'égalité sont de complexité quadratique en n ($O(n^2)$). Le test du vide et la mise sous forme canonique se font en $O(n^3)$.

Etant donné un ensemble d'inéquations D, notons $[\![D]\!]$ l'ensemble des solutions de D.

Définition 1.4. *Deux classes $C_1 = (M_1, D_1)$ et $C_2 = (M_2, D_2)$ sont égales si $M_1 = M_2$ et $[\![D_1]\!] = [\![D_2]\!]$.*

Pour vérifier l'égalité entre deux domaines, les ensembles d'inéquations sont mis sous forme canonique et la vérification est ensuite effectuée sur ces formes canoniques. Cette méthode est plus efficace que la résolution des systèmes d'inéquations et la comparaison, *a posteriori*, de leurs solutions.

Définition 1.5. *Soit $C = (M, D)$ une classe d'états. Une transition t_i est dite tirable (ce qu'on note $t_i \in firable(C)$), depuis C ssi, dans le domaine D, $\alpha(t_i) \leq \min_{t_j \in enabled(M)} \beta(t_j)$.*

Définition 1.6. *Un échéancier de tir d'un SETPN est une séquence finie ou infinie de couples de transitions et de valeurs temporelles $\sigma = \sigma_1, \sigma_2, \sigma_3, \ldots$ avec $\sigma_i = (\delta_i, t_i)$ où t_i sont les transitions et $\delta_i \in \mathbb{R}$ sont les instants de tir.*
Etant donné un échancier de tir fini σ, nous pouvons définir la fonction $time_i(\sigma) = \sum_{k=0}^{i} \delta_k$.

Etant donnée une classe $C = (M, D)$ et une transition tirable t_f, la classe $C' = (M', D')$ obtenue à partir de C par le tir de t_f est donnée par le calcul suivant :

- Nous calculons le nouveau marquage de la même manière que pour des réseaux de Petri classiques : $M' = M - pre(t_f) + post(t_f)$
- D' est calculée de la manière suivante :
  1. Nous effectuons les changements de variables : $\forall i \neq f, x_i \leftarrow x'_i + x_f$ avec $x'_i \geq 0$.
  2. Nous éliminons toutes les variables se rapportant à des transitions désensibilisées par le tir de t_f (en utilisant par exemple la méthode de Fourier-Motzkin [Dan63]),
  3. Nous ajoutons les inéquations relatives aux transitions nouvellement sensibilisées :

$$\forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq x'_k \leq \beta(t_k)$$

  4. Nous calculons une forme canonique du nouveau domaine en utilisant par exemple l'algorithme de Floyd-Warshall [Ber01].

Remarque 1.1. *Les changements de variables modélisent l'écoulement du temps pour les transitions sensibilisées par un changement d'origine du temps : la nouvelle origine correspond à l'instant de tir de t_f.*

Les contraintes $x'_i \geq 0$ peuvent également être écrites $x_i \geq x_f$, ce qui exprime le fait que nous avons choisi de tirer t_f, i.e. *toutes les autres transitions sensibilisées seront tirées plus tard.*

Une fois définie cette méthode d'obtention des successeurs d'une classe, le calcul de l'espace d'états d'un réseau de Petri temporel consiste simplement en la construction classique du graphe d'accessibilité des classes d'états. Cela signifie que, partant de la classe d'états initiale, tous les successeurs obtenus en tirant les transitions tirables sont calculés de manière itérative jusqu'à ce que tous les successeurs produits aient déjà été générés.

1.3.2 Cas des SETPN

Analyse exacte à l'aide de polyèdres

Le passage aux SETPN n'est pas immédiat. En effet, étant donnée la sémantique des SETPN introduite dans la définition 1.2, les domaines des classes d'états ne peuvent pas être calculés de la même manière que pour les réseaux de Petri temporels classiques. Des changements ont ainsi été proposés dans [RD01] afin de permettre leur construction.

Plus précisément, le changement de variables dans le domaine de tir n'est maintenant effectuée plus que pour les *transitions actives*. De plus, au moment de déterminer les transitions tirables, c'est le *domaine de tir actif* qui doit être considéré, *i.e.* le domaine de tir restreint aux variables relatives à des transitions actives.

Si une classe d'états est toujours définie par un marquage et un domaine, la forme générale du domaine n'est, elle, pas préservée. La nouvelle forme générale est celle d'un polyèdre avec des contraintes impliquant jusqu'à n variables, où n est le nombre de transitions sensibilisées par le marquage de la classe :

$$\begin{cases} \alpha_i \leq \theta_i \leq \beta_i, \forall t_i \in enabled(M), a_{i_1}\theta_{i_1} + \cdots + a_{i_n}\theta_{i_n} \leq \gamma_{i_1 \ldots i_n}, \\ \forall (t_{i_1}, \ldots, t_{i_n}) \in enabled(M)^n \text{ et } (a_{i_0}, \ldots, a_{i_n}) \in \mathbb{Z}^n. \end{cases}$$

Exemple de calcul exact

Nous abordons ici l'exemple du calcul exact du domaine d'une classe. Nous partons d'une classe dont le domaine a la même forme que celui d'une classe d'un réseau de Petri temporel standard afin de montrer comment des contraintes additionnelles, que nous dirons « proprement polyédrales », apparaissent.

Soit $C = (M, D)$ une classe d'états du SETPN telle que :

$$D = \begin{cases} \alpha_1 \leq x_1 \leq \beta_1, \\ \alpha_2 \leq x_2 \leq \beta_2, \\ \alpha_3 \leq x_3 \leq \beta_3, \\ \alpha_4 \leq x_4 \leq \beta_4, \\ -\gamma_{21} \leq x_1 - x_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq x_1 - x_3 \leq \gamma_{13}, \\ -\gamma_{41} \leq x_1 - x_4 \leq \gamma_{14}, \\ -\gamma_{32} \leq x_2 - x_3 \leq \gamma_{23}, \\ -\gamma_{42} \leq x_2 - x_4 \leq \gamma_{24}, \\ -\gamma_{43} \leq x_3 - x_4 \leq \gamma_{34} \end{cases} \tag{1.1}$$

Nous supposons aussi que t_1, t_2 et t_3 (auxquelles sont associées les variables x_1, x_2, x_3) sont actives, et que t_4 ne l'est pas. De plus, nous supposons que t_1 est tirable et nous calculons le domaine de la classe obtenue en tirant t_1. La première étape consiste à effectuer le changement de variables $x_i \leftarrow x'_i + x_1$ (en ajoutant les contraintes $x'_i \geq 0$) pour toutes les transitions actives t_i à l'exception de la transition désensibilisée, t_1. Le domaine devient alors :

$$\begin{cases} \alpha_1 \leq x_1 \leq \beta_1, \\ \alpha_2 \leq x_1 + x'_2 \leq \beta_2, \\ \alpha_3 \leq x_1 + x'_3 \leq \beta_3, \\ \alpha_4 \leq x'_4 \leq \beta_4, \\ -\gamma_{21} \leq x_1 - x_1 - x'_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq x_1 - x_1 - x'_3 \leq \gamma_{13}, \\ -\gamma_{41} \leq x_1 - x_4 \leq \gamma_{14}, \\ -\gamma_{32} \leq x_1 + x'_2 - x_1 - x'_3 \leq \gamma_{23}, \\ -\gamma_{42} \leq x_1 + x'_2 - x_4 \leq \gamma_{24}, \\ -\gamma_{43} \leq x_1 + x'_3 - x_4 \leq \gamma_{34} \end{cases} \tag{1.2}$$

L'étape suivante consiste à éliminer la variable x_1. Pour ce faire, nous utilisons la méthode de Fourier-Motzkin. Commençons par réécrire les inéquations de la manière suivante :

$$\begin{cases} \alpha_1 \leq x_1, & x_1 \leq \beta_1, \\ \alpha_2 - x'_2 \leq x_1, & x_1 \leq \beta_2 - x'_2, \\ \alpha_3 - x'_3 \leq x_1, & x_1 \leq \beta_3 - x'_3, \\ -\gamma_{41} + x_4, \leq x_1 & x_1 \leq \gamma_{14} + x_4, \\ -\gamma_{42} + x_4 - x'_2 \leq x_1, & x_1 \leq \gamma_{24} + x_4 - x'_2, \\ -\gamma_{43} + x_4 - x'_3 \leq x_1, & x_1 \leq \gamma_{34} + x_4 - x'_3, \\ \alpha_4 \leq x'_4 \leq \beta_4, \\ -\gamma_{32} \leq x'_2 - x'_3 \leq \gamma_{23}, \\ -\gamma_{21} \leq -x'_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq -x'_3 \leq \gamma_{13} \end{cases} \tag{1.3}$$

La méthode de Fourier-Motzkin consiste alors à écrire que le système admet des solutions si et seulement si les minorants de x_1 sont inférieurs ou égaux à ses majorants. Le système obtenu est alors équivalent au système précédent. Après quelques simplifications, nous obtenons :

$$\begin{cases} max(0, -\gamma_{12}) \leq x'_2 \leq \gamma_{21}, \\ max(0, -\gamma_{13}) \leq x'_3 \leq \gamma_{31}, \\ \alpha_4 \leq x_4 \leq \beta_4, \\ -\gamma_{32} \leq x'_2 - x'_3 \leq \gamma_{23}, \\ -\gamma_{42} - \beta_1 \leq x'_2 - x_4 \leq \gamma_{24} - \alpha_1, \\ -\gamma_{43} - \beta_1 \leq x'_3 - x_4 \leq \gamma_{34} - \alpha_1 \\ \alpha_2 - \gamma_{14} \leq x'_2 + x_4 \leq \beta_2 + \gamma_{41}, \\ \alpha_3 - \gamma_{14} \leq x'_3 + x_4 \leq \beta_3 + \gamma_{41}, \\ \alpha_2 - \gamma_{34} \leq x'_2 + x_4 - x'_3 \leq \beta_2 + \gamma_{43}, \\ \alpha_3 - \gamma_{24} \leq x'_3 + x_4 - x'_2 \leq \beta_3 + \gamma_{42} \end{cases} \tag{1.4}$$

La dernière étape consiste à calculer la forme canonique du domaine. Elle n'est pas détaillée ici car elle est propre aux algorithmes de manipulations des polyèdres. Nous pouvons constater que les huit inéquations obtenues sur les quatre dernières lignes ne peuvent pas être exprimées à l'aide d'une DBM. Elles constituent des inéquations que nous appellerons « proprement polyédrales ». De plus, nous pouvons facilement voir que ces nouvelles inéquations pourront donner des inéquations encore plus complexes (*i.e.* mettant en jeu encore plus de variables) lors du tir d'une nouvelle transition pour le domaine obtenu.

Surapproximation

La manipulation de polyèdres est généralement très coûteuse en temps de calcul. Afin que les algorithmes restent efficaces dans le cas des SETPN, LIME et ROUX ont proposé une méthode de surapproximation des polyèdres à l'aide de DBM [LR03a]. Elle consiste à éliminer du domaine toutes les inéquations ne pouvant pas être représentées par des DBM.

Illustrons cette surapproximation sur l'exemple précédent : elle revient à écrire que le domaine de tir obtenu en fin de calcul peut être approché par le domaine :

$$\begin{cases} max(0, -\gamma_{12}) \leq x'_2 \leq \gamma_{21}, \\ max(0, -\gamma_{13}) \leq x'_3 \leq \gamma_{31}, \\ \alpha_4 \leq x_4 \leq \beta_4, \\ -\gamma_{32} \leq x'_2 - x'_3 \leq \gamma_{23}, \\ -\gamma_{42} - \beta_1 \leq x'_2 - x_4 \leq \gamma_{24} - \alpha_1, \\ -\gamma_{43} - \beta_1 \leq x'_3 - x_4 \leq \gamma_{34} - \alpha_1 \end{cases} \tag{1.5}$$

De manière évidente, cette surapproximation ajoute des états aux classes manipulées, des états qui ne devraient pas être accessibles.

Toutefois, pour la vérification de propriétés de sûreté, la surapproximation n'est pas un problème majeur : puisque nous souhaitons vérifier que rien de « mauvais » ne survient, cette vérification peut être effectuée sur un ensemble d'états qui contient le véritable espace d'états du SETPN. Il subsiste, bien sûr, un risque d'être pessimiste par rapport au système réel.

1.4 Surapproximation et calcul exact

Nous en venons légitimement à nous interroger sur l'origine des cas tels que le graphe des classes obtenu par la méthode de surapproximation est différent de celui généré par le calcul exact. Nous avons donc étudié les paramètres faisant apparaître ces formes polyédrales si particulières. Sur ce sujet, nous sommes parvenus au théorème suivant :

Théorème 1.2. *La surapproximation d'un domaine de tir D' par la forme DBM associée relaxe des contraintes si et seulement si la classe parente $C = (M, D)$ contient à la fois une transition active t_i et une transition suspendue (*i.e. *sensibilisée mais non-active) t_j qui restent sensibilisées lors du tir de la transition t_f et si l'une au moins des deux inéquations suivantes est satisfaite :*

$$\beta_i + \gamma_{jf} < \beta_j + \gamma_{if} \tag{1.6}$$

$$\alpha_j - \gamma_{fi} < \alpha_i - \gamma_{fj} \tag{1.7}$$

Démonstration 1.2. *Soit une classe $C' = (M', D')$. On note $C = (M, D)$ sa classe parente. Pour les besoins de la démonstration, nous supposons que le domaine D correspond au domaine le plus simple possible, c'est-à-dire un domaine sous forme de DBM. Nous supposons par ailleurs que, dans la classe C, trois transitions sensibilisées t_1, t_2 et t_3, interviennent : t_1 et t_2 sont supposées actives, t_3 non. Nous supposons de plus que le tir de t_1 depuis la classe C conduit en C'. Nous effectuons la démonstration dans cette configuration minimale.*

Le domaine D initial, mis sous forme canonique, a la forme suivante :

$$D = \begin{cases} \alpha_1 \leq x_1 \leq \beta_1, \\ \alpha_2 \leq x_2 \leq \beta_2, \\ \alpha_3 \leq x_3 \leq \beta_3, \\ -\gamma_{21} \leq x_1 - x_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq x_1 - x_3 \leq \gamma_{13}, \\ -\gamma_{32} \leq x_2 - x_3 \leq \gamma_{23} \end{cases} \tag{1.8}$$

Nous supposons enfin que l'une au moins des deux inéquations suivantes est satisfaite :

$$\beta_2 + \gamma_{31} < \beta_3 + \gamma_{21}$$

$$\alpha_3 - \gamma_{12} < \alpha_2 - \gamma_{13}$$

Calculons désormais le domaine D' obtenu à partir de D en tirant t_1 : nous commençons par effectuer le changement de variable $x_i \leftarrow x'_i + x_1$ pour toutes les transitions actives à l'exception de la transition désensibilisée, t_1. Cela signifie que nous n'effectuons ce changement que pour la variable x_2 : $x_2 \leftarrow x'_2 + x_1$. Dans la foulée, nous écrivons les inéquations de sorte à pouvoir utiliser la méthode de Fourier-Motzkin. Nous obtenons alors :

$$\begin{cases} \alpha_1 \leq x_1, & x_1 \leq \beta_1, \\ \alpha_2 - x'_2 \leq x_1, & x_1 \leq \beta_2 - x'_2, \\ -\gamma_{31} + x_3, \leq x_1, & x_1 \leq \gamma_{13} + x_3, \\ -\gamma_{32} + x_3 - x'_2 \leq x_1, & x_1 \leq \gamma_{23} + x_3 - x'_2, \\ \alpha_3 \leq x_3 \leq \beta_3, & \\ -\gamma_{21} \leq -x'_2 \leq \gamma_{12} & \end{cases} \tag{1.9}$$

Or le système admet des solutions si et seulement si les bornes inférieures de x_1 sont inférieures ou égales aux bornes supérieures. Nous voyons immédiatement qu'une forme polyédrale apparaît en combinant $\alpha_2 - x'_2 \leq x_1$ et $x_1 \leq \gamma_{13} + x_3$. De même en combinant $x_1 \leq \beta_2 - x'_2$ avec $-\gamma_{31} + x_3 \leq x_1$. Il en résulte $\alpha_2 - \gamma_{13} \leq x'_2 + x_3 \leq \beta_2 + \gamma_{31}$. Par ailleurs, nous pouvons tirer du domaine précédent la liste suivante de contraintes portant sur x'_2 et x_3 :

$$\begin{cases} -\gamma_{12} \leq x'_2 \leq \gamma_{21}, \\ \alpha_2 - \beta_1 \leq x'_2 \leq \beta_2 - \alpha_1, \\ -\gamma_{32} - \gamma_{13} \leq x'_2 \leq \gamma_{23} + \gamma_{31}, \\ \alpha_3 \leq x_3 \leq \beta_3, \\ \alpha_1 - \gamma_{13} \leq x_3 \leq \beta_1 + \gamma_{31}, \\ \alpha_2 - \gamma_{23} \leq x_3 \leq \beta_2 + \gamma_{32} \end{cases} \tag{1.10}$$

Etant donnée la forme canonique du système initial, nous avons $\forall (i,j), \gamma_{ij} \leq \beta_i - \alpha_j$, $\beta_i \leq \beta_j + \gamma_{ij}$ et $\alpha_i - \gamma_{ij} \leq \alpha_j$ et $\forall (i,j,k) \gamma_{ij} \leq \gamma_{ik} + \gamma_{kj}$. Nous en déduisons que le système précédent se simplifie en :

$$\begin{cases} -\gamma_{12} \leq x'_2 \leq \gamma_{21}, \\ \alpha_3 \leq x_3 \leq \beta_3 \end{cases} \tag{1.11}$$

Il reste à montrer que, parmi les deux contraintes polyédrales mises en évidence précédemment, $\alpha_2 - \gamma_{13} \leq x'_2 + x_3 \leq \beta_2 + \gamma_{31}$, au moins une n'est pas redondante avec les contraintes sur $x'_2 + x_3$ que nous pouvons déduire des contraintes individuelles portant sur x'_2 et x_3, à savoir :

$$\alpha_3 - \gamma_{12} \leq x'_2 + x_3 \leq \beta_3 + \gamma_{21} \tag{1.12}$$

Cette vérification est immédiate au vu de l'hypothèse de départ.
Ainsi, le système finalement obtenu fait bien apparaître une forme « proprement polyédrale » qui n'est pas redondante avec les autres contraintes. La surapproximation avec une DBM de ce domaine relaxera donc cette contrainte.

Il reste à démontrer l'implication inverse. Pour ce faire, nous raisonnons par contraposée et nous supposons que la classe parente n'inclut pas à la fois une transition active et une transition suspendue telles que ces deux transitions restent sensibilisées par le tir de t_f (auquel cas il est immédiat que le domaine surapproximé est égal au domaine obtenu par le calcul exact) ou qu'aucune des inéquations portant sur $\alpha_i - \gamma_f j$ et $\beta_i + \gamma_j f$ n'est vérifiée (dans ce deuxième cas, cela signifique que la contrainte polyédrale éventuellement obtenue est redondante avec les contraintes obtenues séparément sur x_i et x_j, autrement dit nous n'avons pas affaire à une véritable forme polyédrale). Mais alors la surapproximation à l'aide des DBM ne relaxe aucune contrainte. Le résultat est ainsi bien démontré.

1.5 Résultat sur le langage des SETPN

Théorème 1.3. *Etant donné un échéancier $\sigma = (\delta_1, t_1), (\delta_2, t_2), \ldots$ avec des états $s_i = (m_i, \nu_i)$, il est possible de construire un chemin $\pi = (M_0, D_0) \xrightarrow{t_1} (M_1, D_1) \xrightarrow{t_2} \ldots$ tel que $\forall i \geq 0, m_i = M_i$.*

Démonstration 1.3. *Cette démonstration est une adaptation de la preuve donnée, pour le théorème 1, par* Yoneda et al. *dans [YS97].*
Classiquement, lors du calcul des graphe des classes, nous commençons par effectuer le changement de variable $x_i \to x_i + x_f$ où t_f représente la transition tirée. Nous éliminons ensuite les variables associées aux transitions désensibilisées - et, avec elles, les inéquations associées. Ces étapes correspondent à des compositions de projections et de changements d'origine et reviennent à raisonner sur des variables temporelles relatives. Pour les besoins de notre démonstration, nous décidons de conserver ici les inéquations relatives aux transitions désensibilisées. Ceci ne modifie pas le polyèdre sur lequel nous raisonnons.

Nous allons construire une fonction $eval_i(t, \sigma)$ qui, étant donné un échéancier σ, donne pour chaque état $s_i = (m_i, \nu_i)$ une valeur à chacune des variables x apparaissant dans D_i de sorte que cette valeur est une solution de D_i.

Deux sortes de variables apparaissent dans D_i : des variables pour les temps d'occurrence des transitions sensibilisées par le marquage m et des variables pour les temps d'occurence des transitions non-sensibilisées dans le marquage m. Ces dernières transitions sont des transitions qui ont été tirées précédemment.

Définissons donc une fonction qui, à chaque variable x_j du domaine des contraintes D_i :

$$eval_i(t_j, \sigma) = \begin{cases} max(time_i(\sigma) + \alpha_j - \nu_i(t, \sigma), time_k(\sigma)) & si\ t_j \in active(m) \\ max(time_n(\sigma) + \alpha_j - \nu_n(t, \sigma), time_k(\sigma)) & si\ t_j \in (enabled(m) - active(m)) \\ time_l(\sigma) & sinon \end{cases}$$

où k est le plus petit indice supérieur ou égal à i tel que t_j est désensibilisée dans l'état (m_k, ν_k), $l \leq i$ est l'indice correspondant à l'état dans lequel t_j a été tirée (est devenue désensibilisée) et $0 \leq n \leq i$ est le maximum de l'indice correspondant à l'état dans lequel t_j était active pour la dernière fois et l'indice correspondant à l'état dans lequel t_j est devenue sensibilisée. Notons que $time_k(\sigma) \leq time_i(\sigma) + \beta_j - \nu_i(x_j, \sigma)$ car t_j est désensibilisée avant que $\nu(t_j)$ n'atteigne β_j. Par ailleurs, posons : $\forall i, eval_i(\bar{0}, \sigma) = 0$.

Pour une transition sensibilisée, $eval_i(t, \sigma)$ représente le temps auquel t sera tirée ou le temps auquel elle aurait pu être tirée (si elle est désensibilisée avant le tir).

Nous allons maintenant montrer qu'étant donné un échéancier σ, nous pouvons construire un chemin π tel que $eval_i(\sigma)$ est une solution de D_i, i.e. *la règle de tir respecte la sémantique de l'échéancier de tir.*

Etape d'initiation ($i = 0$) : $\forall t_j \in enabled(m_0)$, nous avons $eval_0(t_j, \sigma) = max(time_0(\sigma) + \alpha_j - \nu_0(t_j, \sigma), time_k(\sigma)) = max(0 + \alpha_j - 0, time_k(\sigma)) \geq \alpha_j$
Par ailleurs, $\alpha_j \leq \beta_j$ et $time_k(\sigma) \leq \beta_j$ (quelle que soit l'exécution, une transition sensibilisée sera désensibilisée avant que ne soit atteinte sa date de tir au plus tard). Donc $eval_0(t_j, \sigma) \leq \beta_j$. Ainsi toutes les inéquations de D_0 sont bien vérifiées.

Etape d'induction : supposons que $m_i = M_i$ et que $eval_i$ satisfait toutes les inéquations de D_i. Soit (m_{i+1}, ν_{i+1}) l'état obtenu à partir de (m_i, ν_i) par le tir de t_f.
Commençons par montrer que $t_f \in firable(M_i, D_i)$. Puisque $t_f \in enabled(m_i)$ et $t_f \notin enabled(m_{i+1})$, nous avons $eval_i(t_f, \sigma) = time_{i+1}(\sigma)$. Pour toutes les autres transitions sensibilisées dans (m_i, ν_i), i.e. *telles que $t \in enabled(m_i)$, nous avons $eval_i(t) \geq time_{i+1}(\sigma)$. Par conséquent, $eval_i$ satisfait $D_i \cup \{t_f \leq t | t \in enabled(m_i)\}$, de sorte que $t_f \in fireable(m_i)$.*

Soit D_{i+1} le domaine obtenu à partir de D_i par le tir de t_f. Nous devons désormais montrer que $eval_{i+1}$ est une solution de D_{i+1}. Pour ce faire, nous devons distinguer trois cas :

- *La transition t_j était sensibilisée dans l'état (m_i, ν_i) et le reste dans l'état (m_{i+1}, ν_{i+1}). Il faut alors distinguer deux sous-cas :*
  - *La transition t_j est active dans l'état (m_{i+1}, ν_{i+1}). Nous avons alors :*

$$\begin{aligned} eval_{i+1}(t_j, \sigma) &= max(time_{i+1}(\sigma) + \alpha_j - \nu_{i+1}(\sigma), time_k(\sigma)) \\ &= max(time_i(\sigma) + \delta_i + \alpha_j - (\nu_{i+1}(\sigma) + \delta_i), time_k(\sigma)) \\ &= max(time_i(\sigma) + \alpha_j - \nu_i(\sigma), time_k(\sigma)) \\ &= eval_i(t_j, \sigma) \end{aligned}$$

  - *La transition t_j n'est pas active dans l'état (m_{i+1}, ν_{i+1}). Nous avons alors, de manière immédiate, $eval_{i+1}(t_j, \sigma) = eval_i(t_j, \sigma)$.*
    Ainsi $\forall t_j \in enabled(m_i) \cap enabled(m_{i+1}), eval_{i+1}(t_j, \sigma) = eval_i(t_i, \sigma)$. Comme nous ne changeons pas la contrainte portant sur les transitions qui restent sensibilisées, $eval_{i+1}(t_j, \sigma)$ est une solution de D_{i+1}.

- *La transition t_j est nouvellement sensibilisée. Nous avons alors $\nu_{i+1}(t_j, \sigma) = 0$, d'où $eval_{i+1}(t_j, \sigma) = max(time_{i+1}(\sigma) + \alpha_j, time_k(\sigma))$. Ainsi, $eval_{i+1}(t_j, \sigma) \geq \alpha_j$. Par ailleurs, $time_k(\sigma) \leq_{i+1} (\sigma) + \beta_j$ et $time_{i+1}(\sigma) + \alpha_j \leq time_{i+1}(\sigma) + \beta_j$, d'où $eval_{i+1}(t_j, \sigma) \leq time_{i+1}(\sigma) + \beta_j$. Finalement, nous avons bien $\alpha_j \leq eval_{i+1} - time_{i+1}(\sigma) \leq \beta_j$.*

- *La transition t_j a déjà été tirée : $\forall t_j \in disabled(m_i), eval_{i+1}(t_j, \sigma) = eval_i(t_j, \sigma)$. Comme nous ne changeons pas les contraintes portant sur les transitions qui ont été désensibilisées, $eval_{i+1}(t_j, \sigma)$ vérifie les inéquations de D_{i+1}.*

Finalement, nous avons bien démontré que $eval_{i+1}$ satisfait D_{i+1}. L'assertion est donc bien reconduite au pas suivant.

Théorème 1.4. *Etant donné un chemin $\pi = (M_0, D_0) \xrightarrow{t_1} (M_1, D_1) \xrightarrow{t_2} \ldots$, il est possible de construire un échéancier $\sigma = (\delta_1, t_1), (\delta_2, t_2), \ldots$ avec des états $s_i = (m_i, \nu_i)$ tel que $\forall i \geq 0, m_i = M_i$.*

Démonstration 1.4. *Supposons que nous ayions un chemin de classes d'états $\pi = (M_0, D_0) \xrightarrow{t_1} (M_1, D_1) \xrightarrow{t_2} \ldots$ avec une solution pour chaque domaine D_i. Définissons une valuation ν_i telle que $\forall t_j, \nu_0(t_j) = 0$ et :*

$$\nu_{i+1}(t_j) = \begin{cases} 0 & \text{si } t_j \text{ est nouvellement sensibilisée dans la marquage } m_i. \\ \nu_i(t_j) + clock_{i+1} - clock_i & \text{sinon} \end{cases}$$

Avec $clock_0 = 0$ et $clock_{i+1}$ est une valeur associée à t_{i+1} dans la solution de D_{i+1}. Il est alors immédiat de montrer que l'échéancier $\sigma = (\delta_1, t_1), (\delta_2, t_2), \ldots$ constitue une exécution valide du réseau.

Des deux résultats précédents, nous déduisons le théorème suivant.

Théorème 1.5. *Pour un SETPN, l'ensemble des mots du graphe des classes est identique à l'ensemble des mots abstrait temporellement du réseau.*

1.6 Algorithmes de calcul du graphe des classes pour les SETPN

Nous ne présenterons ici que les algorithmes propres aux SETPN. Le principe de calcul du graphe des classes des SETPN est similaire au calcul effectué pour les réseaux de Petri temporels. Les différences résident dans les méthodes utilisées pour obtenir la liste des transitions tirables et pour déterminer la classe suivante.

i : **entier** ;
j : **entier** ;
PolyèdreTest : **Polyèdre** ;
MatriceTest : **Matrice de rationnels** ;
FirableTransitions : **Liste de transitions** ;
Initialiser la liste des transitions tirables FirableTransitions à la liste vide
Initialiser le polyèdre PolyèdreTest à un polyèdre vide de dimension égale à la dimension du domaine de tir de la classe considérée
Initialiser la matrice MatriceTest à une matrice vide
Calculer le marquage actif de la classe courante
Déterminer les transitions actives à partir du marquage actif
Pour toutes les transitions actives t_i **faire**
  Copier dans MatriceTest la matrice des contraintes du domaine de tir de la classe considérée
  Pour toutes les transitions actives $t_j, j \neq i$ autres que la transition active courante **faire**
    Ajouter à MatriceTest la contrainte $x_i \leq x_j$
  Fin Pour
  Construire le polyèdre PolyèdreTest associé à la matrice des contraintes MatriceTest
  Si (PolyèdreTest est non-vide) **Alors**
    Ajouter t_i à la liste des transitions tirables FirableTransitions
  Fin Si
  Retourner FirableTransitions
Fin Pour

Algorithme 1: Méthode de détermination des transitions tirables (paramètre d'entrée : la classe courante $C = (M, D)$; résultat : la liste des transitions tirables FirableTransitions)

i : **entier** ;
NewPolyhedron : **Polyèdre** ;
IntermediatePolyhedron : **Polyèdre** ;
NewMarking : **Vecteur d'entiers** ;
NewlyEnabledTransitions : **Liste de transitions** ;
DisabledTransitions : **Liste de transitions** ;
Initialiser la matrice des contraintes du domaine de la classe suivante NewConstraintsMatrix à une matrice vide
Initialiser le polyèdre intermédiaire IntermediatePolyhedron au polyèdre de la classe courante
Calculer le marquage actif de la classe courante
Déterminer la liste des transitions sensibilisée de la classe courante PreviouslyEnabledTransitions à partir du marquage de la classe courante
Calculer le marquage de la classe suivante : NewMarking = M $-pre(t_f) + post(t_f)$
Calculer le marquage actif de la classe suivante
Déterminer la liste NewlyEnabledTransitions des transitions nouvellement sensibilisées de la classe suivante
Déterminer la liste DisabledTransitions des transitions désensibilisées (celles qui sont sensibilisées dans la classe courante et qui ne le sont plus dans la classe suivante)
Pour toutes les transitions nouvellement sensibilisées t_i **faire**
    Ajouter à NewPolyhedron les contraintes relatives à la transition nouvellement sensibilisée considérée : $\alpha_i \leq x_i \leq \beta_i$
Fin Pour
Pour toutes les transitions actives t_i de la classe courante, qui n'appartiennent pas à DisabledTransitions **faire**
    Effectuer dans IntermediatePolyhedron le changement de variable $x_i \leftarrow x'_i + x_f$
Fin Pour
Pour toutes les transitions désensibilisées $t_i \in DisabledTransitions$ **faire**
    Réécrire toutes les inéquations de IntermediatePolyhedron dans lesquelles x_i intervient de sorte à faire apparaître ses bornes inférieures et supérieures
    Ajouter dans NewPolyhedron toutes les contraintes de IntermediatePolyhedron dans lesquelles x_i n'intervient pas
    Ajouter dans NewPolyhedron les contraintes suivantes : toutes les bornes inférieures de x_i doivent être inférieures aux bornes supérieures de x_i
Fin Pour
Minimiser le polyèdre NewPolyhedron obtenu
Retourner le couple (NewMarking,NewPolyhedron) qui définit la classe suivante

Algorithme 2: Méthode de détermination de la classe suivante (paramètre d'entrée : la classe courante $C = (M, D)$ et la transition tirée t_f ; résultat : la classe suivante $C' = (M', D')$)

Il s'agit désormais d'implémenter ces algorithmes en C++ au sein de l'outil d'analyse des réseaux de Petri temporels qu'est ROMEO. Pour ce faire, nous avons besoin de recourir à une bibliothèque externe de manipulation des polyèdres. C'est ce travail que nous allons décrire dans le chapitre suivant.

Chapitre 2

Implémentation

L'équipe « temps réel » de l'IRCCyN a développé Romeo, un outil pour l'analyse des réseaux de Petri [RLG04].

Romeo est composé d'une interface graphique écrite en Tcl/Tk et de deux modules de calculs de l'espace d'états des réseaux de Petri temporels :
- Gpn2, basé sur la méthode du graphe des classes d'états [BD91] ;
- Mercutio, basé sur la méthode du graphe des régions [AD94].

Romeo permet d'exécuter :
- Le calcul du graphe des classe d'états d'un réseau de Petri temporel [BD91] ;
- Le calcul de l'automate (temporisé) des classes d'un réseau de Petri temporel [LR03b] ;
- Le calcul de l'automate (temporisé) des marquages d'un réseau de Petri temporel [GRR05] ;
- Le calcul de l'automate à classes d'états d'un réseau de Petri temporel étendu à l'ordonnancement en utilisant une surapproximation [LR04] ;
- La traduction structurelle d'un réseau de Petri temporel vers un automate temporisé [CR04] ;
- La surapproximation du graphe des classes des réseaux de Petri temporels étendus à l'ordonnancement [LR03a] ;
- La vérification à la volée de propriétés sur les marquages accessibles.

Romeo n'effectue que le calcul approché du graphe des classes des SETPN. Il s'agit donc d'implémenter, dans Gpn2, les algorithmes de calcul exact du graphe des classes des réseaux de Petri temporels étendus à l'ordonnancement. Pour ce faire, nous devons recourir à une bibliothèque externe de manipulation des polyèdres.

Nous allons donc brièvement rappeler quelques éléments théoriques sur les polyèdres. Nous enchaînerons avec la présentation de quelques-unes des plus importantes bibliothèques de manipulation de ces objets pour, au final, arrêter notre choix sur l'une de celles-ci.

2.1 Brefs rappels sur la théorie sur les polyèdres

Définition 2.1. *Un polyèdre convexe rationnel est un ensemble de points (dans un espace vectoriel de dimension n) qui satisfait un nombre fini d'inéquations linéaires à coefficients rationnels.*

Définition 2.2. *Un ensemble $P \subseteq \mathbb{R}^n$ est un polyèdre non nécessairement fermé, ou NNC (pour* Not Necessarily Closed*), si et seulement si P peut s'exprimer comme l'intersection d'un nombre fini de demi-espaces affines (ouverts ou fermés) de $\mathbb{R}^n$ ou $n = 0$ et $P = \emptyset$.*

Notation 2.1. *Notons $\mathbb{P}_n$ l'ensemble des polyèdres convexes de $\mathbb{R}^n$.*

Définition 2.3. *Un polyèdre $P \in \mathbb{P}_n$ est borné s'il existe $\lambda \in \mathbb{R}^+$ tel que $P \subseteq \{x \in \mathbb{R}^n | -\lambda \leq x_j \leq \lambda, j = 0 \ldots n-1\}$. Un polyèdre borné est un polytope.*

Les polyèdres NNC possèdent deux représentations duales : ils peuvent être définis soit comme un ensemble de contraintes linéaires, soit comme un ensemble de générateurs.

Représentation à l'aide de contraintes

Définition 2.4. *Par définition, un polyèdre P de $\mathbb{P}_n$ est un ensemble de solutions d'un système contraint. Avec une notation matricielle, nous avons :*

$$P = \{x \in \mathbb{R}^n | A_1 x = b_1, A_2 x \geq b_2, A_3 x > b_3\}$$

avec $A_i \in \mathbb{R}^{m_i} \times \mathbb{R}^{\kappa}$, $b_i \in \mathbb{R}^{m_i}$ où $m_1, m_2, m_3 \in \mathbb{N}$ correspondent respectivement au nombre d'équations, au nombre d'inéquations non-strictes et au nombre d'inéquations strictes.

Définition 2.5. *Etant donnés une matrice $A \in \mathbb{R}^{m \times n}$ et un vecteur $b \in \mathbb{R}^m$, un couple (b, A) est une* H-représentation *d'un polyèdre convexe NNC P ssi $P = \{x \in \mathbb{R}^n | b + Ax \geq 0\}$.*

Représentation à l'aide de générateurs

Définition 2.6. *Soit $S = \{x_1, \ldots, x_k\} \subseteq \mathbb{R}^n$ un ensemble fini de vecteurs. $v = \sum_{j=1}^{k} \lambda_j x_j$, combinaison linéaire de vecteurs de S, est dite :*

- *positive (ou conique) si $\forall j \in \{1, \ldots, k\}, \lambda_j \in \mathbb{R}^+$*
- *affine si $\sum_{j=1}^{k} \lambda_j = 1$*
- *une combinaison convexe si elle est à la fois positive et affine.*

Notation 2.2. *Notons $linear.hull(S)$ (respectivement $conic.hull(S)$, $affine.hull(S)$, $convex.hull(S)$) l'ensemble des combinaisons* linéaires *(respectivement* positives, affines, convexes*) de vecteurs de S.*

Définition 2.7. *Soient $P, C \subseteq \mathbb{R}^n$ tels que $P \cup C = S$ $nnc.hull(P, C)$ représente l'ensemble des combinaisons convexes de vecteurs de S tels qu'il existe un j tel que $\lambda_j > 0$.*

Cela signifie qu'il existe un vecteur de P qui joue un rôle actif dans la combinaison convexe.

Remarque 2.1. *$linear.hull(S)$ est un espace affine, $conic.hull(S)$ un cône topologiquement fermé, $convex.hull(S)$ un polytope topologiquement fermé et $nnc.hull(P, C)$ un polytope NNC.*

Définition 2.8. *Soit $P \in \mathbb{P}_n$ un polyèdre NNC.*

- *Un vecteur $p \in P$ est un point de P ;*
- *un vecteur $c \in \mathbb{R}^n$ est un point de fermeture de P s'il appartient à la fermeture topologique de P.*
- *Un vecteur $r \in \mathbb{R}^n$ avec $r \neq 0$, est un rayon (un rayon ou une direction infinie) de P si $P \neq \emptyset$ et $p + \lambda r \in P$ pour tout point $p \in P$ et $\lambda \in \mathbb{R}^+$.*
- *Un vecteur $l \in \mathbb{R}^n$ est une droite de P si l et $-l$ sont tous deux des rayons de P.*

Théorème 2.1. *Tout polyèdre NNC $P \in \mathbb{P}_n$ peut être représenté par un ensemble fini de droites L, de rayons R, de points P et de points de fermeture C de P.*

Le quadruplet $\mathcal{G} = (L, R, P, C)$ est dit système générateur *de P au sens où :*

$$P = linear.hull(L) + conic.hull(R) + nnc.hull(P, C)$$

où $+$ représente la somme de Minkowski.

Définition 2.9. *Etant données deux matrices $V \in \mathbb{R}^{p \times n}$ et $R \in \mathbb{R}^{q \times n}$, un couple (V, R) est une* V-représentation *d'un polyèdre convexe NNC P ssi $P = conv(V) + cone(R)$ où $conv(M)$ (respectivement $cone(M)$) représente l'enveloppe convexe (respectivement l'enveloppe positive) des vecteurs lignes de la matrice M.*

Représentations minimales

Définition 2.10. *Un système de contraintes $\mathcal{C}$ d'un polyèdre NNC $\mathcal{P}$ est dit minimal si aucun sous-ensemble de $\mathcal{C}$ n'est un système de contraintes de $\mathcal{P}$.*

De manière similaire, un système générateur $\mathcal{G} = (L, R, P, C)$ d'un polyèdre NNC $\mathcal{P}$ est dit minimal s'il n'existe pas de système générateur $\mathcal{G}' = (L', R', P', C') \neq \mathcal{G}$ de $\mathcal{P}$ tel que $L' \subseteq L, R' \subseteq R, P' \subseteq P$ et $C' \subseteq C$.

Définition 2.11. *Une H-représentation est dite équivalente à une V-représentation si les polyèdres qu'elles représentent sont identiques.*

Théorème 2.2. *Tout polyèdre convexe NNC admet une V-représentation et une H-représentation canoniques minimales [AFP02].*

Une représentation canonique et minimale d'un polyèdre s'obtient en supprimant les redondances de systèmes d'inéquations et en effectuant des transformations affines.

Théorème 2.3. *La suppression de redondances, le calcul d'une représentation canonique et la programmation linéaire sont des problèmes polynomialement équivalents [AFP02].*

La complexité de la vérification d'égalité entre un H-polyèdre P_H et un V-polyèdre P_V est inconnue. Cette vérification comporte deux parties : « est-ce que $P_V \subseteq P_H$ » et « est-ce que $P_H \subseteq P_V$ »?

Il est facile de répondre à la première question en vérifiant simplement que chaque générateur de P_V satisfait la H-représentation de P_H.

Le deuxième problème est, au moins, co-NP complet [FO85]. Toutefois, cette complexité est inconnue quand la première question admet une réponse positive.

Il existe des méthodes pour passer d'une représentation à l'autre et obtenir des représentations minimisées en enlevant les contraintes (ou générateurs) redondants.

2.2 Algorithmes de calcul de l'espace d'états à l'aide de polyèdres

2.3 Présentation des différentes bibliothèques

Il existe un certain nombre de bibliothèques écrite en C/C++ pour la manipulation de polyèdres. Nous présenterons ici les principales. Elles possèdent toutes un certain nombre de points communs. D'une part, au niveau des opérations qu'elles permettent d'exécuter sur les polyèdres. D'autre part, dans leur représentation des polyèdres, basée sur la technique dite de double description : elle recourent toutes, pour manipuler les resprésentations de ces polyèdres, à une implémentation des algorithmes de Chernikova [Che65]. Pour stocker les données propres à la définition des polyèdres, ces bibliothèques utilisent des matrices de coefficients. Au rang des différences entre les bibliothèques, nous pouvons citer les choix de représentation sur les entiers.

Signalons qu'HyTech, l'outil de vérification développé au sein de l'Université de Cornell [HHWT95], dispose de sa propre bibliothèque de manipulation de polyèdres.

2.3.1 Polka

Historiquement, Polka est une des premières bibliothèques de manipulation de polyèdres à avoir vu le jour, sous l'égide de Nicolas Halbwachs *et al.* [HPR97] au sein de VERIMAG. A l'heure actuelle, Polka n'est plus une bibliothèque intéressante au sens où elle a été supplantée par ses successeurs. De plus, les sources de Polka ne sont pas disponibles, la bibliothèque n'étant pas placée sous licence de logiciel libre.

2.3.2 Polylib

Polylib a été originellement conçue en 1993 au sein de l'IRISA par Doran WILDE et Hervé LEVERGE [Wil93]. Elle se base sur la méthode de description duale de Motzkin ainsi que sur une implémentation de l'algorithme de Chernikova écrite par Hervé LeVerge. La bibliothèque a été écrite en C. Aujourd'hui, tout comme Polka, Polylib n'est plus intéressante car elle a été dépassée par les bibliothèques auxquelles elle a donné naissance.

2.3.3 PolyLib

PolyLib est une évolution de la bibliothèque Polylib initialement développée par Doran WILDE et Hervé LEVERGE [Loe99]. Elle bénéficie de mises à jour régulières mais présente moins de fonctionnalités que les bibliothèques New Polka et Parma Polyhedra Library : contrairement à ces dernières, elle ne permet pas de manipuler des polyèdres NNC, c'est-à-dire représentés avec des contraintes strictes telle que $x > y$.

2.3.4 New Polka

New Polka est une bibliothèque maintenue par Bertrand JEANNET, pour la manipulation des polyèdres convexes dont les contraintes et les générateurs sont exprimés à l'aide de coefficients rationnels [Jea02]. Elle est codée en C de sorte qu'elle peut être utilisée dans des programmes développés aussi bien en C qu'en C++. Elle est basée sur Polylib et sur Polka. Son développement a été motivé par le besoin de manipuler des entiers codés sur 64 bits. La gestion de la mémoire a été améliorée et les matrices de saturation peuvent être conservées en mémoire, économisant ainsi du temps de calcul. De plus, New Polka permet de gérer des polyèdres NNC. New Polka possède l'avantage d'être mis à disposition des utilisateurs sous licence libre.

2.3.5 Parma Polyhedra Library

La Parma Polyhedra Library (PPL) est une bibliothèque de manipulation de polyèdres basée sur New Polka, développée sous l'impulsion de Roberto BAGNARA *et al.* [BRZH02]. Elle s'en différencie en ce qu'elle est écrite en C++. La PPL fut la première bibliothèque permettant de manipuler des polyèdres convexes NNC, cette partie de New Polka étant longtemps restée incomplète. La PPL est plus ergonomique que les autres bibliothèques en ce qu'elle permet la définition de contraintes en écrivant simplement les inéquations sous la forme $x+2*y+5*z <= 7$. Elle est entièrement dynamique.

Dernier avantage, et non des moindres, elle est distribuée sous licence GNU GPL.

Elle semble toutefois être maintenue avec moins de régularité que New Polka, ce qui nous a fait préférer cette dernière pour l'implémentation que nous avions à effectuer.

2.4 Utilisation de New Polka

New Polka est une bibliothèque écrite en C. Elle permet d'effectuer les principales opérations dont nous avons besoin pour manipuler les domaines de tirs associés aux classes des SETPN.

Pour éviter tout problème de débordement, New Polka manipule des entiers multi-précision. Il en résulte toutefois une perte de performance en terme de vitesse. Une option, à la compilation, permet de préciser à quel type d'entiers l'utilisateur souhaite recourir :

- entiers multi-précision : c'est le bibliothèque *GNU Multi Precision library* (GMP) qui est alors utilisée ;
- entiers standards tels que manipulés par la machine (codés sur 32 ou 64 bits) avec, toutefois, aucune vérification portant sur un débordement éventuel ;
- entiers `long long int`.

Comme nous allons le voir, les égalités bénéficient d'un traitement à part : il existe des méthodes qui permettrent des les minimiser plus efficacement qu'en les considérant comme deux inégalités opposées. De même, New Polka fait la distinction entre les générateurs bidirectionnels et les rayons normaux.

Le format des objets dépend de la manière dont la bibliothèque a été initialisée (en activant l'option permettant de manipuler des contraintes strictes ou non). Dans ce qui suit, la dimension des polyèdres sera notée n.

Si l'option `strict` n'a pas été activée :

- $[0, b, a_0, \ldots, a_{n-1}]$ représente la contrainte d'égalité $a_0x_0 + \ldots + a_{d-1}x_{d-1} + b = 0$;
- $[1, b, a_0, \ldots, a_{n-1}]$ représente la contrainte d'inégalité $a_0x_0 + \ldots + a_{d-1}x_{d-1} + b \geq 0$;
- $[0, 0, a_0, \ldots, a_{n-1}]$ représente la droite de direction $(a_0, \ldots, a_{d-1})$;
- $[1, 0, a_0, \ldots, a_{n-1}]$ représente le rayon de direction $(a_0, \ldots, a_{d-1})$;
- $[1, b, a_0, \ldots, a_{n-1}]$ représente le sommet $(a_0/b, \ldots, a_{d-1}/b)$;
- $[d, b, a_0, \ldots, a_{n-1}]$ représente l'expression affine $x \mapsto (a_0/d)x_0 + \ldots + (a_{d-1}/d)x_{d-1} + b/d$.

La nature d'un vecteur (contrainte, générateur ou expression affine) est inférée en fonction du contexte.

Pour utiliser des inégalités strictes, nous ajoutons un élément dans le vecteur, entre la deuxième et la troisième position. Pour plus de détails, le lecteur pourra se référer au manuel de New Polka, fourni avec le code source de la bibliothèque.

New Polka permet d'effectuer, entre autres, les opérations suivantes sur les polyèdres :

- Calcul de la réprésentation minimale du polyèdre. Une fois cette minimisation effectuée, les deux types de représentation du polyèdre sont disponibles ;
- Obtention de la matrice des contraintes, des générateurs et de la matrice de saturation d'un polyèdre ;
- Obtention de la dimension et du nombre de contraintes d'un polyèdre ;
- Détermination de l'intersection avec un autre polyèdre ;
- Calcul de l'enveloppe convexe de deux polyèdres ;
- Ajout de contraintes ;
- Suppression de dimensions du polyèdre ;
- Test du vide sur le polyèdre ;
- Test de l'inclusion d'un polyèdre dans un autre ;
- Test de l'égalité de deux polyèdres ;
- Substitution de variable à l'intérieur d'un polyèdre ;
- Ajout de dimensions et projection, ...

C'est grâce à New Polka que nous avons implémenté, dans ROMEO, le calcul exact des classes d'états des SETPN[1]. Dans le chapitre suivant, nous allons détailler les principaux résultats que nous avons obtenus après avoir effectué cette implémentation.

[1] Concernant la gestion des contraintes de positivité par New Polka, il nous paraît utile de préciser le point suivant : il convient de mentionner explicitement, lors de la définition du polyèdre, les contraintes $x_i \geq 0$, même si ces contraintes se déduisent implicitement d'autres inéquations (telles que $x_1 \geq 2$). En effet, New Polka considère le polyèdre défini par $(x_1 = 2 et x_2 = 3)$ comme vide ; lorsqu'on ajoute les contraintes de positivité sur les variables x_1 et x_2, le polyèdre défini par $(x_1 = 2, x_2 = 3, x_1 \geq 0, x_2 \geq 0)$ est, lui, bien considéré comme non vide par la bibliothèque.

Chapitre 3

Mise en œuvre

Pour vérifier le bon fonctionnement de notre implémentation, nous l'avons testée sur un certain nombre d'exemples. Dans ce qui suit, nous allons présenter les principaux résultats auxquels nous sommes parvenus. Dans un premier temps, nous nous assurons de la bonne marche de notre programme sur un exemple très simple de deux tâches en concurrence sur un unique processeur. Puis nous illustrerons les différences entre calcul exact et surapproximation à l'aide des DBM à travers un exemple où le graphe des classes approché ne se confond pas avec le graphe des classes exact. Enfin, nous tirerons des conclusions générales portant sur les tests que nous avons menés.

3.1 Exemple de deux tâches en concurrence sur un même processeur

Dans cette partie, nous reprenons le réseau de Petri temporel étendu à l'ordonnancement de la figure 1.1.

La classe initiale est la suivante :

$$\begin{cases} \{P_1, P_3\}, \\ \begin{cases} 2 \leq x_1 \leq 3, \\ 1 \leq x_3 \leq 4, \end{cases} \end{cases}$$

Les tâches associées aux places P_1 et P_3 sont toutes deux attachées au processeur $\gamma = 1$. Mais la priorité de la place P_3 est supérieure. De ce fait, des transitions T_1 et T_3, seule la transition T_3 est active. Depuis la classe initiale, nous ne pouvons donc tirer que la transition T_3.

Par le tir de T_3 depuis la classe initiale, nous obtenons la classe C_1 définie de la manière suivante :

$$\begin{cases} \{P_1, P_4\}, \\ \begin{cases} 2 \leq x_1 \leq 3, \\ 3 \leq x_4 \leq 5, \end{cases} \end{cases}$$

La place P_4 a une priorité supérieure à celle de la place P_1. La seule transition active (et tirable) depuis la classe C_1 est donc T_4.

Par le tir de T_3 depuis la classe initiale, nous obtenons la classe C_1 définie de la manière suivante :

$$\begin{cases} \{P_1\}, \\ \begin{cases} 2 \leq x_1 \leq 3, \end{cases} \end{cases}$$

La seule transition sensibilisée et tirable est alors la transition T_1. De C_2, nous obtenons la classe C_3 en tirant T_1 :

$$\begin{cases} \{P_2\}, \\ \{ \ 1 \leq x_2 \leq 2, \end{cases}$$

Dans la classe C_3, la seule transition sensibilisée et tirable est T_2. Nous obtenons alors la classe C_4 :

$$\begin{cases} \{\emptyset\}, \\ \{ \text{ Domaine de tir vide} \end{cases}$$

Il ne reste plus aucune transition tirable. L'algorithme est arrivé à sa terminaison.

Dans cet exemple, nous voyons qu'il n'apparaît jamais de forme proprement polyédrale de type $\alpha \leq x_i + x_j \leq \beta$. Le résultat obtenu via la surapproximation correspond donc au résultat calculé à partir de l'algorithme exact. C'est bien ce que nous vérifions en pratique en utilisant la méthode que nous avons implémentée dans ROMEO. Le graphe des classe retourné par le calcul exact à base de polyèdres correspond exactement à celui obtenu via la surapproximation. Le fichier renvoyé par ROMEO est le suivant :

C0 – T 3 → C1

C1 – T 4 → C2

C2 – T 1 → C3

C3 – T 2 → C4

Class C0
Marking
1 (0) P 1
0 (0) P 2
1 (1) P 3
0 (0) P 4
Firing Domain
$2 \leq$ T 1
T 1 ≤ 3
$1 \leq$ T 3
T 3 ≤ 4
$0 \leq$ T 1
$0 \leq$ T 3

Class C1
Marking
1 (0) P 1
0 (0) P 2
0 (0) P 3
1 (1) P 4
Firing Domain
T 4 ≤ 5
$3 \leq$ T 4

$2 \leq$ T 1
T 1 ≤ 3

Class C2
Marking
1 (1) P 1
0 (0) P 2
0 (0) P 3
0 (0) P 4
Firing Domain
$2 \leq$ T 1
T 1 ≤ 3

Class C3
Marking
0 (0) P 1
1 (1) P 2
0 (0) P 3
0 (0) P 4
Firing Domain
T 2 ≤ 2
$1 \leq$ T 2

Class C4
Marking
0 (0) P 1
0 (0) P 2
0 (0) P 3
0 (0) P 4
Firing Domain

3.2 Exemple de réseau où le calcul exact est nécessaire

3.2.1 Cas où le calcul exact comporte moins de classes que la surapproximation

Etudions à présent le SETPN de la figure 3.1. Nous allons illustrer, sur ce cas, le fait que la surapproximation ajoute bien des états par rapport au calcul exact, et qu'elle n'est donc pas toujours suffisante dès lors que nous souhaitons avoir des résultats pointus sur la vérification de certains systèmes.

Considérons la séquence de tir t_4, t_1, t_5. Le calcul approché obtenu via l'algorithme qui était déjà implémenté dans GPN2 nous indique que cette séquence conduit à la classe suivante :

$$\begin{cases} \{p_2, p_3, p_6\}, \\ \begin{cases} 0 \leq x_2 \leq 4, \\ 0 \leq x_3 \leq 4, \\ 4 \leq x_6 \leq 4 \end{cases} \end{cases}$$

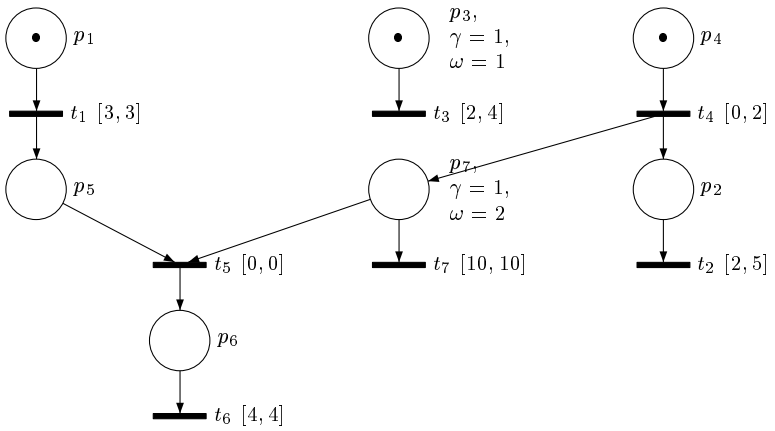

FIG. 3.1 – Un réseau de Petri temporel étendu à l'ordonnancement

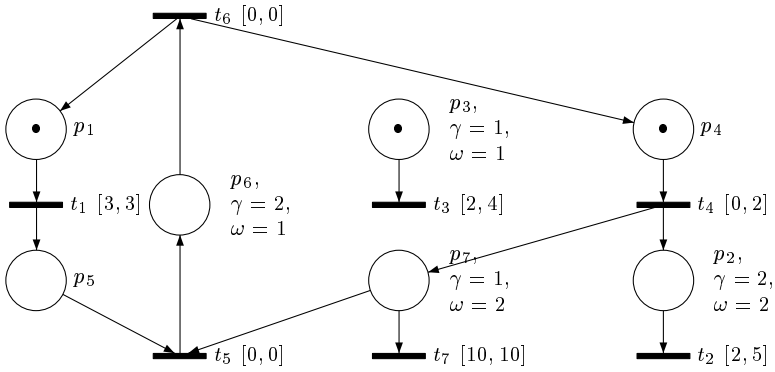

FIG. 3.2 – SETPN pour lequel le calcul exact retourne plus de classe que la méthode de surapproximation

Le calcul exact, lui, fournit la classe définie par :

$$\begin{cases} \{p_2, p_3, p_6\}, \\ \begin{cases} 0 \leq x_2 \leq 4, \\ 0 \leq x_3 \leq 4, \\ 4 \leq x_6 \leq 4, \\ 1 \leq x_2 + x_3 \leq 6 \end{cases} \end{cases}$$

Alors que, dans la classe obtenue via la surapproximation, la transition t_6 est tirable, elle ne l'est pas dans la classe calculée par la méthode exacte. En effet, cela impliquerait $x_2 = x_3 = x_6 = 4$, mais alors $x_2 + x_3$ serait égal à 8, ce qui est supérieur à 6. La surapproximation ajoute donc, entre autres, les classes obtenues par le tir de t_6.

Sur cet exemple, le calcul exact donne 18 classes et 25 transitions alors que la surapproximation retourne 21 classes et 31 transitions. Nous allons maintenant adapter cet exemple pour montrer que, dans certains cas, le calcul exact retourne plus de classes que la surapproximation.

3.2.2 Cas où le calcul exact comporte plus de classes que la surapproximation

Envisageons maintenant le réseau de Petri temporel étendu à l'ordonnancement de la figure 3.5. Dans ce cas, l'implémentation exacte retourne 36 classes alors que la surapproximation n'en

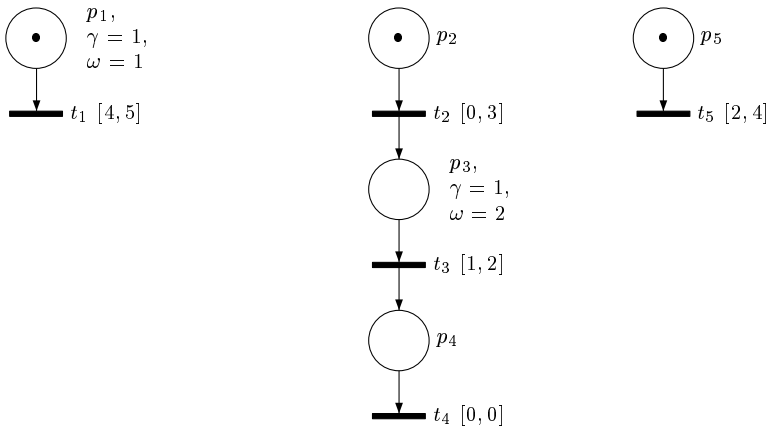

FIG. 3.3 – Un SETPN dont un seul domaine présente une forme polyédrale

retourne que 32. Cet exemple illustre le fait que certaines classes obtenues via la surapproximation englobent plusieurs classes d'états du véritable graphe des classes du SETPN. Lors de l'analyse des résultats obtenus via le calcul approché et via le calcul exact, il convient de prêter attention à chacune des classes rencontrées et non pas seulement au nombre total de classes. En effet, il est possible de construire des exemples pour lesquels la surapproximation et le calcul exact aboutissent au même nombre de classes d'états et de transitions, mais où l'une de ces classes fait apparaître une forme polyédrale. Cette forme polyédrale disparaît ensuite dans le calcul des classes suivantes, mais elle peut avoir son importance dès lors que nous désirons effectuer une vérification fine du système.

Le calcul exact et le calcul surapproximé retournent, pour le réseau de la figure 3.3, 12 classes et 14 transitions. Mais en étudiant attentivement les fichiers de résultat, nous découvrons que la classe obtenue par la séquence de tir $t_2.t_5$ présente en fait une forme purement polyédrale. La classe obtenue est en effet la suivante :

$$\left\{ \begin{array}{l} \{p_1, p_3\}, \\ \left\{ \begin{array}{l} 1 \leq x_1 \leq 5, \\ 0 \leq x_3 \leq 2, \\ 1 \leq x_1 + x_3 \leq 5 \end{array} \right. \end{array} \right.$$

Cette forme polyédrale n'a pas d'impact sur le calcul des classes suivantes car elle disparaît lors du tir de la transition suivante, à savoir t_3, seule transition active dans la classe considérée. Un pareil cas illustre le fait que le calcul exact et la surapproximation peuvent conduire au même nombre de classes et de transitions, mais avec des classes qui ne sont pas nécessairement intrinsèquement identiques.

3.3 Exemple de SETPN admettant un nombre infini de classes

Nous allons désormais étudier le réseau de la figure 3.4, réseau proposé par Bernard BERTHOMIEU. Il est possible de montrer, par récurrence, que pour tout entier $k \geq 0$, la séquence de tir $(T_4.T_1.T_5.T_2.T_3)^k$ est tirable depuis la classe initiale et conduit en des classes toutes différentes. Définissons la classe C_k par :

$$\left\{ \begin{array}{l} \{p_3, 2 * p_4, p_5, p_9\}, \\ \left\{ \begin{array}{l} t_3 = 1, \\ 2^k x_1 \leq 2^k + 1, \\ 0 \leq x_4 \\ 1 \leq x_1 - x_4 \end{array} \right. \end{array} \right.$$

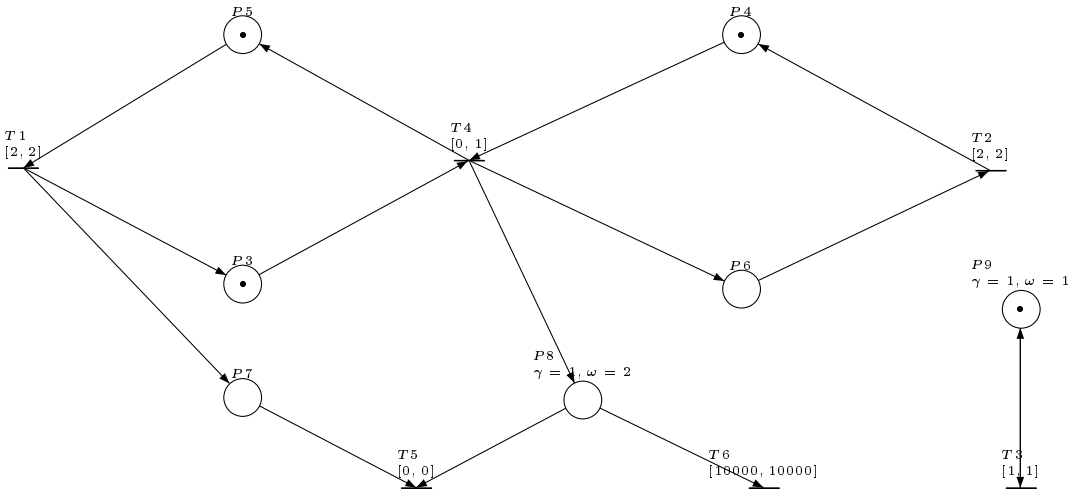

FIG. 3.4 – Un réseau de Petri temporel étendu à l'ordonnancement admettant un nombre infini de classes

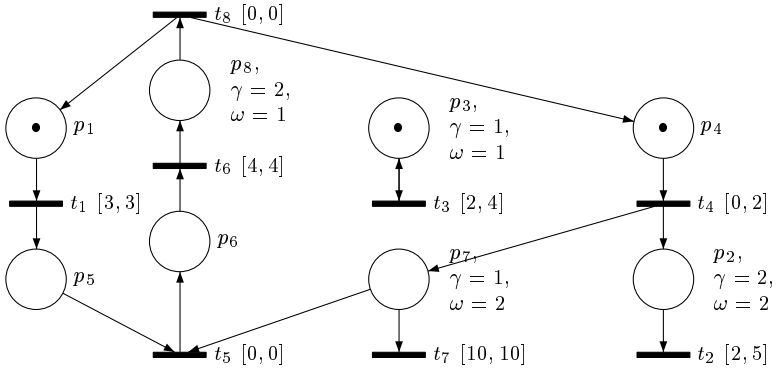

FIG. 3.5 – Un autre réseau de Petri temporel étendu à l'ordonnancement admettant un nombre infini de classes

On montre que :

- La séquence de tir $(T_4.T_1.T_5.T_2.T_3)$ ne change pas le marquage : le vecteur marquage est alors $M^t = (1, 2, 1, 0, 0, 0, 1)$.
- Pour $k \geq 1$, $(T_4.T_1.T_5.T_2.T_3)$ est tirable depuis C_k et conduit en C_{k+1}.

La suite de rationnels $(2^{k+1}/2^k)_{k\geq 1}$ est infinie ; par conséquent, les classes de la suite $(C_k)_{k\geq 1}$ sont toutes différentes. Géométriquement, le domaine de tir de C_k, pour $k \geq 1$, est un triangle rectangle isocèle. A chaque tir de $(T_4.T_1.T_5.T_2.T_3)$, la longueur de ses deux côtés égaux diminue de moitié.

Nous avons testé ce réseau dans notre implémentation du calcul polyédral ainsi que sur TINKA, l'outil de vérification développé au sein du LAAS [1]. Nous obtenons bien les résultats attendus, à savoir que les programmes ne se terminent pas. L'affichage de résultats intermédiaires permet de vérifier que les classes $(C_k)_{k\geq 1}$ ont bien la forme attendue. Ainsi, après quatre tirs de la séquence $(T_4.T_1.T_5.T_2.T_3)$, nous obtenons la classe suivante :

$$\left\{ \begin{array}{l} \{p_3, 2 * p_4, p_5, p_9\}, \\ \left\{ \begin{array}{l} x_3 = 1, \\ 16x_1 \leq 17, \\ 0 \leq x_4 \\ 1 \leq x_1 - x_4 \end{array} \right. \end{array} \right.$$

Ce réseau est d'autant plus intéressant qu'il illustre le fait qu'un SETPN peut être borné mais le nombre de classes infinies. En effet, le réseau est bien borné ($m(P_3) + m(P_5) = 2$ et $m(P_2) + m(P_4) = 2$ constituent deux invariants du réseau tandis que l'inégalité $m(P_7) + m(P_8) \geq 2$ reste vérifiée en permanence) ; mais, comme nous l'avons vu, le nombre de ses classes est infini. Il est possible de démontrer des résultats similaires pour le SETPN de la figure 3.5, adapté du réseau de la figure 3.1.

3.4 Conclusions générales

Pour vérifier le bon fonctionnement de notre implémentation du calcul exact du graphe des classes des SETPN, nous avons comparé nos résultats avec ceux retournés par TINKA, l'outil de vérification développé au sein du LAAS. Ces résultats sont bien identiques. En terme de temps de calcul, notre implémentation est plus lente que celle de TINKA. Cela peut paraître très important. Mais, d'une part, le sujet de ce stage n'était pas de concevoir l'implémentation la plus rapide possible du calcul polyédral (il aurait fallu, pour cela, avoir des connaissances beaucoup plus poussées en C++ et, éventuellement, recourir à une bibliothèque plus performante que New Polka). D'autre part, ROMEO fait actuellement l'objet d'une nouvelle implémentation visant à en améliorer l'efficacité. Cette implémentation, basée sur la recherche de la généricité maximale, fusionnera en un seul programme les modules GPN et MERCUTIO. Nous profiterons de ce projet pour optimiser notre implémentation du calcul exact.

[1] Tinka n'était pas encore officiellement disponible au moment de notre travail. Nous avons fait les tests sur une version intermédiaire

Conclusion

Au cours de ce stage pratique de DEA, nous avons démontré, dans le cas des réseaux de Petri temporels étendus à l'ordonnancement, l'égalité entre l'ensemble des mots du graphe des classes et le langage temporellement abstrait du réseau. Nous avons par ailleurs proposé un algorithme de calcul, à l'aide de polyèdres, de l'espace d'états de ces réseaux. Le problème de l'accessibilité étant indécidable pour les SETPN, l'algorithme ne se termine pas forcément. Nous l'avons intégré dans Gpn2, le module de calcul du graphe des classes de l'outil Romeo.

Pour ce faire, nous avons eu recours à une bibliothèque externe de manipulation des polyèdres : New Polka. Grâce à elle, nous avons pu minimiser les systèmes d'équations mis en jeu lors du calcul des domaines de tir propres à une classe et les mettre sous une forme canonique ; cela a notamment permis de tester l'égalité entre plusieurs domaines. Ce travail a été effectué en C++, sur la base des classes déjà implémentées dans Gpn2.

Nous avons aussi étudié les critères faisant apparaître, dans le domaine d'une classe d'un SETPN, une forme proprement polyédrale. Nous avons ainsi exhibé une condition nécessaire et suffisante pour déterminer si la surapproximation relaxe, ou non, des contraintes par rapport aux classes d'états obtenues par le calcul exact.

Notre travail porte désormais sur l'amélioration des performances de notre implémentation. Romeo faisant actuellement l'objet d'une nouvelle implémentation afin d'en accroître la généricité, nous comptons profiter de ce projet pour reprendre et améliorer notre implémentation du calcul exact du graphe des classes des réseaux de Petri temporels étendus à l'ordonnancement. Nous espérons ainsi pouvoir réduire les temps de calcul lors de l'utilisation des méthodes polyédrales.

Bibliographie

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[AFP02] D. Avis, K. Fukuda, and S. Picozzi. On canonical representations of convex polyhedra. In A. M. Cohen, X.-S. Gao, and N. Takayama, editors, *Mathematical Software, Proceedings of the First International Congress of Mathematical Software*, pages 350–360. World Scientific Publishing, 2002.

[BD91] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Softw. Eng.*, 17(3) :259–273, 1991.

[Ber01] B. Berthomieu. La méthode des classes d'état pour l'analyse des réseaux temporels. In *3e congrès Modélisation des Systèmes Réactifs (MSR'2001)*, pages 275–290, Toulouse, France, 2001. Hermes Science.

[BRZH02] Roberto Bagnara, Elisa Ricci, Enea Zaffanella, and Patricia M. Hill. Possibly not closed convex polyhedra and the parma polyhedra library. In *Proceedings of the 9th International Symposium on Static Analysis*, pages 213–229. Springer-Verlag, 2002.

[Che65] N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of system of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5(2) :258–263, 1965.

[CR04] Franck Cassez and Olivier (H.) Roux. Structural translation from time Petri nets to timed automata. In *Fourth International Workshop on Automated Verification of Critical Systems (AVoCS'04)*, London (UK), September 2004.

[Dan63] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[Dil90] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 197–212. Springer-Verlag New York, Inc., 1990.

[FO85] R. M. Freund and J. B. Orlin. On the complexity of four polyhedral set containment problems. *Mathematical Programming*, 33 :139–145, 1985.

[GRR05] Guillaume Gardey, Olivier (H.) Roux, and Olivier (F.) Roux. State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems*, 2005. to appear.

[Hen96] Thomas Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 278–292, New Brunswick, New Jersey, 1996.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech : The next generation. In *IEEE Real-Time Systems Symposium*, pages 56–65, 1995.

[HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2) :157–185, August 1997.

[Jea02] B. Jeannet. Convex polyhedra library, release 1.1.3c edition. Available at http ://www.irisa.fr/prive/ Bertrand.Jeannet/newpolka.html, March 2002.

[Loe99] V. Loechner. Polylib : A library for manipulating parameterized polyhedra. Available at http ://icps.u-strasbg.fr/ loechner/polylib/, March 1999.

[LPY95] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In *Proceedings of the 10th International Symposium on Fundamentals of Computation Theory*, pages 62–88. Springer-Verlag, 1995.

[LR03a] Didier Lime and Olivier H. Roux. Expressiveness and analysis of scheduling extended time petri nets. In *Proceedings of the 5th IFAC conference on fieldbus and their applications*. Elsevier Science, 2003.

[LR03b] Didier Lime and Olivier (H.) Roux. State class timed automaton of a time Petri net. In *The 10th International Workshop on Petri Nets and Performance Models, (PNPM'03)*, pages 124–133, Urbana, USA, September 2003. IEEE Computer Society.

[LR04] Didier Lime and Olivier (H.) Roux. A translation based method for the timed analysis of scheduling extended time Petri nets. In *The 25th IEEE International Real-Time Systems Symposium, (RTSS'04)*, Lisbon, Portugal, December 2004. IEEE Computer Society Press.

[Men82] M. Menasche. *Analyse des réseaux de Petri temporisés et application aux systèmes distribués.* Université Paul Sabatier, Toulouse, 1982.

[Mer74] P. M. Merlin. *A Study of the Recoverability of Computing Systems.* Irvine : Univ. California, PhD Thesis, 1974.

[RD01] Olivier H. Roux and Anne-Marie Déplanche. Extension des réseaux de Petri t-temporels pour la modélisation de l'ordonnancement de tâches temps-réel. In *3e congrès Modélisation des Systèmes Réactifs (MSR'2001)*, pages 327–342, Toulouse, France, 2001. Hermes Science.

[RLG04] Olivier Roux, Didier Lime, and Guillaume Gardey. Romeo. Available at http ://www.irccyn.ec-nantes.fr/irccyn/d/en/equipes/TempsReel/logs/software-2-romeo, 2004.

[Wil93] D. K. Wilde. *A library for doing polyhedral operations.* Master's thesis, Oregon State University, Corvallis, Oregon, 1993.

[YS97] Tomohiro Yoneda and Bernd-Holger Schlingloff. Efficient verification of parallel real time systems. *Form. Methods Syst. Des.*, 11(2) :187–215, 1997.