Université de Nantes

École Doctorale

Sciences et Technologies de l'Information et des Matériaux

Année : 2004

Thèse de Doctorat de l'Université de Nantes

Spécialité : Automatique et Informatique Appliquée

Présentée et soutenue publiquement par :

Didier Lime

le 1er Décembre 2004
à l'École Centrale de Nantes

Titre :

VÉRIFICATION D'APPLICATIONS TEMPS RÉEL À L'AIDE DE RÉSEAUX DE PETRI TEMPORELS ÉTENDUS

JURY

Rapporteurs	Serge HADDAD	Professeur de l'Université de Paris-Dauphine
	Guy JUANOLE	Professeur de l'Université Paul Sabatier de Toulouse
Président	Joseph SIFAKIS	Directeur de recherche CNRS - VERIMAG Grenoble
Examinateurs	Bernard BERTHOMIEU	Chargé de recherche CNRS - LAAS Toulouse
	Olivier H. ROUX	Maître de Conférences de l'Université de Nantes
	Yvon TRINQUET	Professeur de l'Université de Nantes
Invité	Jean-Louis FERRIER	Professeur de l'Université d'Angers

Directeur de thèse : Yvon Trinquet
Laboratoire : IRCCyN
Composante de rattachement du directeur de thèse : IUT de Nantes
Co-encadrant : Olivier H. Roux
Laboratoire : IRCCyN
Composante de rattachement du co-encadrant : IUT de Nantes

Thèse n°ED 366-174

Remerciements

J'aimerais, avant tout, remercier Serge Haddad et Guy Juanole pour avoir accepté d'être les rapporteurs de cette thèse, ainsi que Bernard Berthomieu, Joseph Sifakis et Jean-Louis Ferrier pour leur participation à mon jury.

En particulier, je tiens à remercier Bernard Berthomieu pour la collaboration extrêmement enrichissante que nous avons développée.

Merci à Yvon Trinquet pour m'avoir accueilli au sein de l'équipe temps réel, pour sa gentillesse, la confiance qu'il m'a accordée et cette jolie balade en avion !

Merci du fond du cœur à Olivier Roux. La réussite à tout point de vue de cette thèse lui doit énormément. Merci de m'avoir suppléé, à la suite de mon départ au Danemark, auprès des rapporteurs et pour un certain nombre de tâches administratives des plus ingrâtes. Mais surtout, merci pour ces trois formidables années d'enrichissement humain et scientifique. Puissent-elles avoir engendré une amitié durable.

Merci également à toutes les personnes avec qui j'ai travaillé ou que j'ai simplement eu le plaisir de cotôyer pendant ces trois ans, en particulier les membres de l'équipe temps réel et du projet MOVES de l'IRCCyN, les membres du département Mathématiques et Informatique de l'École Centrale de Nantes, les membres du département Informatique de l'École des Mines de Nantes et, bien sûr, tous les thésards de l'IRCCyN.

En particulier, merci à Morgan pour avoir relu ma thèse (sans y avoir été contraint par de quelconques pressions), pour son aide dans la gestion de mes affaires nantaises et pour avoir été le meilleur étudiant de DEA, humainement et scientifiquement, que j'ai encadré jusque là.

Merci également à Mikaël pour la belle réussite de notre vie de bureau commune, et pour tout le reste.

Merci à Pierrick, Aude, Nicolas, Claire, Kristell, Cédric et Félix pour nos inoubliables midis de Rock N' Roll. Merci à Frédéric, Pierre-Emmanuel, Julien, Franck, Yann, Sébastien, Guillaume, Myriam et tous les autres, cités ou non, pour trois ans de bonne humeur auxquels ils ont largement contribué.

Enfin, je tiens à remercier mes parents et toute ma famille pour leur soutien inconditionnel et constant. Merci en particulier à mon frère : il m'a toujours montré la voie et c'est probalement grâce à lui que cette aventure à commencé, il y a 22 ans, lorsqu'il m'a appris à lire.

Aalborg, le 5 janvier 2005
Didier Lime

Table des matières

Première partie

Introduction

Chapitre 1

Introduction

Dans ce chapitre, nous présentons brièvement les systèmes temps réel et certains problèmes liés à leur conception. En particulier, nous abordons les problèmes de l'ordonnancement et de la validation.

1.1 Contexte général

La classe des systèmes *temps réel* regroupe les programmes informatiques qui pilotent une application en réagissant à des stimuli reçus d'un environnement évolutif. Cela leur vaut également le nom de systèmes *réactifs*.

Alors que le nombre d'applications pilotées par des systèmes temps réel augmente fortement depuis plusieurs années, nous pouvons également observer une croissance sur deux autres points importants.

Premièrement, de plus en plus d'applications *critiques*, c'est-à-dire mettant en jeu des vies humaines et - ou - des coûts financiers importants, sont progressivement placées sous le contrôle de systèmes temps réel. Cela implique que la *vérification* du bon fonctionnement de ces systèmes est essentielle. Par ailleurs, il est important de détecter les erreurs le plus en amont possible de façon à minimiser les coûts engendrés par leur correction. Des procédures de validation sont donc particulièrement indiquées lors de la phase de conception.

Deuxièmement la *complexité* des applications concernées est de plus en plus grande. La taille et la complexité des systèmes temps réel les pilotant reflète bien sûr cette évolution. Cela engendre de nouveaux problèmes pour leur conception.

C'est dans ce contexte que se situe cette thèse. Nous y développons de nouvelles techniques pour une vérification plus efficace et de nouveaux modèles permettant de prendre en compte l'évolution de la complexité des systèmes temps réel.

1.2 Systèmes temps réel

Un système temps réel S comporte en général plusieurs processeurs, plusieurs tâches par processeur et un réseau par lequel ces tâches communiquent. La conception de ces systèmes comprend donc au moins trois grandes étapes.

La première est de définir l'*architecture logicielle*, c'est-à-dire définir les différentes tâches logicielles permettant de répondre au cahier des charges.

La deuxième consiste à définir l'*architecture support*. Celle-ci est constituée non seulement du matériel (processeurs, réseaux) mais aussi de supports d'exécution temps réel, ou *exécutif temps réel*, permettant de gérer l'exécution des tâches et leurs accès aux ressources.

La dernière étape consiste à faire la « projection » de l'architecture logicielle sur l'architecture support, ce qui fournit l'*architecture opérationnelle*. Pour cette phase, étant donné un ensemble de propriétés souhaitables ou requises $(P_i)_{i\in\mathbb{N}}$, la problématique est de placer puis d'ordonnancer les tâches sur les processeurs de telle sorte qu'elles vérifient les propriétés P_i. Si cela s'avère impossible, il faut redéfinir les architectures logicielle ou matérielle.

1.3 Ordonnancement

Un processeur est une ressource *critique* dans le sens où une seule tâche peut en disposer à un instant donné. Or, chaque tâche demande le processeur selon une *loi d'activation* qui lui est propre. Celle-ci peut-être *périodique* de période T : une instance de la tâche est activée tous les T unités de temps. Elle peut également être *sporadique* : il existe une durée minimale connue, appelée *délai d'inter-arrivée*, entre deux instances d'une même tâche. Enfin, elle peut être *apériodique*, c'est-à-dire que les instants d'activation des instances de la tâche ne sont pas contraints. Il est donc évident qu'à certains moments, plusieurs tâches peuvent requérir le processeur simultanément.

Le choix de la tâche à laquelle est alloué le processeur à un instant donné est réalisé par un module de l'exécutif, appelé *ordonnanceur*, selon un ensemble de règles appelées *politique d'ordonnancement*.

Pour la mise en œuvre de la politique d'ordonnancement, deux approches principales sont distinguées : l'approche *statique* et l'approche *dynamique*.

1.3.1 Approche statique (hors ligne)

L'approche statique consiste à construire, avant l'exécution du système, un échéancier des exécutions des différentes tâches de telle sorte que le système S vérifie les propriétés P_i. Cet échéancier est ensuite « rejoué » lors de l'exécution du système. Pour des configurations de tâches quelconques, un tel échéancier est *a priori* infini. La mémoire disponible pour le stocker étant elle finie, l'approche statique repose donc sur l'existence d'un cycle dans l'exécution du système, qui est en général défini à partir des activations des tâches périodiques.

L'ordonnancement statique est, par nature, sensible aux changements dans la définition du système : l'ajout de tâches, par exemple, peut nécessiter la reconstruction complète du cycle, ce qui est un problème NP-Difficile. La gestion des tâches *sporadiques* ou *apériodiques* pose également problème avec cette approche. Enfin, l'occupation mémoire du cycle d'exécution peut être problématique dans certains cas.

En contrepartie, l'ordonnanceur utilise très peu le processeur et la satisfaction des propriétés P_i du système est assurée par la synthèse du cycle d'exécution.

1.3.2 Approche dynamique (en ligne)

Dans l'approche dynamique, l'ordonnancement des tâches est déterminé pendant l'exécution du système en fonction d'une politique basée sur l'état courant du système. Dans cette approche, la politique d'ordonnancement prend habituellement la forme de priorités affectées aux tâches, la tâche la plus prioritaire étant exécutée.

Nous pouvons notamment distinguer deux classes de politiques d'ordonnancement : les politiques *préemptives* et les politiques *non préemptives*.

Dans le cas préemptif, si l'ordonnanceur est appelé alors qu'une tâche τ est déjà en cours d'exécution et qu'une tâche plus prioritaire τ' est prête, alors τ est interrompue et τ' est exécutée. La tâche τ est alors dite *préemptée* et son exécution reprendra au point d'interruption lorsqu'elle sera à nouveau la tâche la plus prioritaire.

Dans le cas non préemptif, les tâches ne peuvent pas être interrompues par des tâches plus prioritaires. Celles-ci doivent donc attendre, par exemple, que la tâche en cours d'exécution se termine ou se mette en attente d'une ressource pour être exécutées.

Selon que les priorités des tâches sont déterminées une fois pour toutes ou en fonction de l'état courant du système, nous parlerons respectivement de politiques d'ordonnancement à *priorités fixes* et de politiques d'ordonnancement à *priorités dynamiques*.

Dans le cas de priorités fixes, celles-ci peuvent être fixées empiriquement ou systématiquement, en fonction des paramètres des tâches. Les politiques les plus répandues sont :

- *Rate Monotonic* (RM) [LL73] : la priorité d'une tâche est inversement proportionnelle à sa période (Il s'agit ici d'un raccourci de langage car les priorités sont en pratique des valeurs entières. Il faut donc interpréter cela du point de vue de la relation d'ordre des priorités par : la tâche de plus grande période possède la priorité la plus faible) ;
- *Deadline Monotonic* (DM) [LW82] : la priorité d'une tâche est inversement proportionnelle à son échéance.

En ce qui concerne les politiques à priorités dynamiques, nous pouvons citer :

- *Earliest Deadline First* (EDF) [LL73] : la priorité d'une tâche est inversement proportionnelle au temps restant jusqu'à son échéance ;
- *Least Laxity First* (LLF) [MD78] : la priorité d'une tâche est inversement proportionnelle à sa laxité, c'est à dire au temps restant jusqu'à son échéance moins la charge de travail restante.

Si deux tâches ont même priorité, il faut mettre en œuvre une politique de choix secondaire qui peut-être par exemple de type *First In First Out* (FIFO) ou *Round-Robin*. Dans ce dernier cas, les tâches sont exécutées en *pseudo parallélisme* recevant tour à tour une petite « tranche » d'allocation processeur.

L'ordonnancement dynamique est moins sensible à la variation des paramètres du système et plus adapté pour gérer les tâches apériodiques et sporadiques.

Par contre, le surcoût en temps processeur lié à l'application de la politique d'ordonnancement peut poser problème pour certaines applications. Par ailleurs, la politique étant définie, il faut encore vérifier que le système S vérifie bien les propriétés P_i, la première étant l'ordonnançabilité, c'est-à-dire que les tâches respectent toutes leurs échéances temporelles.

Cette thèse se place dans le contexte de l'approche dynamique.

1.4 Validation

Le cahier des charges d'une application informatique est en général formulé en langage naturel. Il est donc soumis à l'interprétation des rédacteurs de la spécification, ce qui est source d'erreurs. Par ailleurs, pour des programmes d'une taille raisonnable, il est impossible, pour le concepteur, d'appréhender toutes les interactions entre les différents composants [1].

[1] Nous sommes bien là en présence de l'illustration flagrante d'un célèbre morceau de sagesse informatique qui nous rappelle qu'« un programme informatique fait ce qu'on lui dit de faire et non pas ce qu'on veut qu'il

Cela peut conduire à des situations où le programme ne remplit plus sa fonction. Il s'agit alors d'une *défaillance* du système.

Pour détecter ces situations imprévues et problématiques, la méthode la plus courante consiste à réaliser des tests du programme pour une série de situations aléatoires ou identifiées par le concepteur. L'intérêt principal de ces tests est qu'ils permettent de détecter des erreurs sans *a priori* sur une propriété ou un composant en particulier. Par contre, il est impossible d'être exhaustif et de tester tous les cas possibles par cette méthode. Elle est donc indispensable mais insuffisante, comme l'illustre cette remarque de Dijkstra [Dij72] : « Program testing can be used to show the presence of bugs, but never to show their absence ».

Pour améliorer l'efficacité des tests, de nombreux travaux (voir par exemple [BT01]) cherchent à générer automatiquement des scenarii à partir d'une spécification formelle du système. Cette *génération de tests* permet d'obtenir des scenarii plus ciblés sur des composants du système identifiés comme source potentielle d'erreurs.

Une autre approche, la *démonstration automatique*, consiste à appliquer automatiquement des règles de déduction logique sur une spécification formelle du système afin de prouver une propriété exprimée dans le même langage (voir par exemple [Rus01]).

Enfin, plutôt que de déduire logiquement les propriétés de la spécification, il est possible de les vérifier en explorant l'espace des états du système, éventuellement de façon exhaustive. Cela conduit à des méthodes dites de *vérification* ou encore de *model-checking*. C'est dans ce cadre que ce situe cette thèse, nous allons détailler cette approche par la suite.

Ces trois dernières approches appartiennent à la famille des « méthodes formelles » car elles s'appuient sur une description mathématique de la spécification du système : un « modèle formel ».

---

fasse ».

Chapitre 2

Vérification des systèmes temps réel

Dans ce chapitre, nous présentons un aperçu des techniques de vérification des systèmes temps réel. Dans un premier temps nous nous intéressons à l'étude analytique de l'ordonnançabilité. Cette approche présentant un certain nombre de limites, nous présentons ensuite les modèles et techniques dits formels. Dans un premiers temps nous abordons les modèles temporisés permettant la prise en compte de synchronisations complexes et de l'écoulement du temps. Dans un second temps, nous nous intéressons au problème spécifique de la prise en compte de l'ordonnancement par les modèles formels.

2.1 Introduction

La première étape de la vérification d'un système informatique temps réel est sa modélisation. Étant donnée une question posée sur le système, un modèle peut être défini comme une abstraction du système telle que la réponse à la question est la même pour le système et son modèle. Cela implique qu'un modèle n'est valable que pour un ensemble de propriétés à vérifier et ne peut refléter complètement le comportement réel du système. Par ailleurs, il y a en général une opposition entre l'*expressivité* d'un modèle, c'est-à-dire sa capacité à représenter un nombre important de caractéristiques du système, et sa *simplicité* en termes de vérification, c'est-à-dire la décidabilité des problèmes liés à son analyse et la complexité algorithmique de cette analyse.

La seconde étape est la conception d'algorithmes de vérification de propriétés sur ces modèles. Dans [Lam77], Lamport classe les propriétés en deux catégories principales : les propriétés de *vivacité* qui assurent que sous certaines conditions, quelque chose finira par avoir lieu et les propriétés de sûreté qui expriment le fait que quelque chose de « mauvais » ne se produira jamais. Ces propriétés peuvent être vérifiées en explorant l'espace d'états du modèle, totalement ou en partie.

Cette exploration est cependant soumise au problème bien connu de l' *explosion combinatoire* : la taille de l'espace d'états augmente très rapidement avec la taille du système. D'un point de vue théorique, les algorithmes de vérification ont donc en général une complexité assez élevée et, pour un certain nombre de modèles, leur terminaison est indécidable. Cependant, en pratique, bon nombre de systèmes peuvent être vérifiés avec succès.

2.2 Approches analytiques de l'ordonnançabilité en ligne

Étant donné un modèle de tâche et une politique d'ordonnancement, de nombreux travaux s'attachent à définir des critères mathématiques d'ordonnançabilité d'ensembles de tâches. Dans cette section, nous donnerons les idées principales relatives à l'approche analytique de l'ordonnançabilité. Pour une étude plus complète et plus détaillée, le lecteur pourra se référer par exemple à [Hla04].

Considérons le modèle simple de la définition 1.

Définition 1 (Modèle des tâches) *Une tâche τ_i est un quintuplet $(T_i, C_i, D_i, J_i, O_i)$ où :*
- *T_i est la période de la tâche i*
- *C_i est le temps d'exécution de la tâche i sans préemption*
- *D_i est l'échéance de la tâche i*
- *J_i est la gigue d'activation de la tâche i*
- *O_i est l'offset de la tâche i*

Dans ce modèle, toutes les tâches sont périodiques. O_i représente le décalage du démarrage de la tâche τ_i par rapport à l'origine des temps et J_i représente un intervalle de temps dans lequel la tâche peut se réveiller : J_i modélise une incertitude sur chaque activation de la tâche τ_i.

Soit E un ensemble de n tâches. Les tâches sont dites *synchrones* si $\forall i \in [\![1, n]\!], O_i = 0$. Elles sont dites *à échéances sur requêtes* si $\forall i \in [\![1, n]\!], D_i = C_i$.

Pour la politique d'ordonnancement *Rate Monotonic* (RM), Liu et Layland donnent une condition suffisante d'ordonnançabilité de cet ensemble dans le cas où les tâches sont synchrones, sans gigue d'activation et à échéances sur requêtes [LL73] :

$$\sum_i \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{2.1}$$

$\sum_i \frac{C_i}{T_i}$ est appelé le *facteur d'utilisation* et représente le taux d'utilisation du processeur.

Sous les mêmes hypothèses, et pour la politique d'ordonnancement *Earliest Deadline First* (EDF), Liu et Layland donnent une condition nécessaire et suffisante d'ordonnançabilité de l'ensemble E :

$$\sum_i \frac{C_i}{T_i} \leq 1 \tag{2.2}$$

Des travaux s'attachent à définir de tels critères d'ordonnançabilité pour des modèles moins restrictifs (par exemple [Cof76, BMR90]). Cependant, les critères produits sont parfois peu pratiques à utiliser et les modèles considérés restent restreints.

Une autre approche pour évaluer l'ordonnançabilité d'un ensemble de tâches consiste à s'appuyer sur la condition nécessaire et suffisante suivante :

$$E \text{ est ordonnançable } \Leftrightarrow R_i \leq D_i \tag{2.3}$$

Où R_i est le temps de réponse de la tâche (ou de l'instance de la tâche) τ_i, c'est-à-dire le temps entre l'activation de la tâche et son achèvement. Grâce à cette condition, le problème de l'ordonnançabilité se réduit alors au problème du calcul des temps de réponse des tâches.

2.2.1 Évaluation des temps de réponse

Tâches indépendantes

Pour un ensemble de tâches indépendantes synchrones et sans gigues d'activation, nous avons [LL73] :

$$R_i = C_i + \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{C_i}{T_j} \right\rceil C_j \tag{2.4}$$

hp(i) est l'ensemble des tâches de plus haute priorité que la tâche i et $\lceil x \rceil$ représente le plus petit entier supérieur à x. Le terme entre $\lceil . \rceil$ est donc le nombre de fois où la tâche i sera préemptée par la tâche j.

Sous les conditions énoncées, les temps de réponse obtenus sont exacts, cependant ces conditions sont très restrictives.

L'analyse peut être facilement étendue pour prendre en compte la gigue. L'équation précédente devient

$$\begin{aligned} w_i &= C_i + \textstyle\sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j \\ R_i &= w_i + J_i \end{aligned} \tag{2.5}$$

Cependant, cette formule fournit des majorants des temps de réponse : il s'agit donc d'un calcul « pessimiste » car le système peut être déclaré non ordonnançable par ce calcul alors qu'il l'est en réalité.

L'idée sous-jacente à cette formule consiste à calculer le temps de réponse de la tâche en considérant le pire cas où toutes les tâches sont réveillées en même temps avec des gigues maximales et les invocations suivantes se font avec des gigues nulles. C'est un scenario qui n'existe pas toujours, le calcul est donc *pessimiste*.

Ces résultats ont été étendus pour obtenir des temps de réponse *exacts* pour des modèles moins restrictifs avec gigues, *offsets*, durées d'exécution variables[1]... (par exemple [Tin94, Hla04]) à l'aide d'algorithmes de complexité polynomiale en le nombre de tâches. Cependant ces résultats ne concernent que des tâches indépendantes.

Tâches communicantes

Dans le cas où les tâches se synchronisent entre elles, par exemple par l'utilisation d'événements ou de sémaphores, ou par des messages dans le cas multiprocesseur, nous ne disposons pas de calcul exact des temps de réponse. Une méthode classique pour étudier l'ordonnançabilité d'un ensembles de tâches avec précédences consiste à modéliser les précédences par des *offsets* et des gigues d'activation définies à partir des meilleurs et pires temps de réponse des tâches en amont des chaînes de précédence [TC94, PH98, HKR01]. Soient deux tâches synchrones et sans gigues d'activation τ_1 et τ_2. Supposons par exemple que la tâche τ_2 doive attendre la fin de la tâche τ_1 avant de s'exécuter ; il y a donc un lien de précédence entre τ_1 et τ_2. Étant donnés le le pire temps de réponse $R^+(\tau_1)$ de τ_1 ainsi que son meilleur temps de réponse[2] $R^-(\tau_1)$, il est clair que la tâche τ_2 ne pourra pas s'exécuter avant au moins $R^-(\tau_1)$ unités de temps et sera retardée au plus de $R^+(\tau_1)$ unités de temps. Le système avec précédences

[1] Pour des tâches indépendantes, le pire cas est obtenu lorsque les durées d'exécution des tâches sont maximales.

[2] Obtenu par une méthode similaire au calcul des pire temps de réponse.

$E = \{\tau_1, \tau_2\}$ peut être transformé en un système de tâches indépendantes $E' = \{\tau'_1, \tau'_2\}$ où $\tau'_1 = \tau_1$ et τ'_2 est la tâche obtenue en ajoutant à la tâche τ_2 l'*offset* $O'_2 = R^-(\tau_1)$ et la gigue d'activation $J'_2 = R^+(\tau_1) - R^-(\tau_1)$. L'une des méthodes exactes sur les ensembles de tâches indépendantes peut alors être appliquée à E'. La méthode est ensuite itérée par point fixe.

Cependant, si les temps de réponse obtenus sont exacts pour l'ensemble E', ils sont, dans le cas général pessimistes pour l'ensemble E. C'est d'autant plus vrai, que dans le cas de tâches non indépendantes, les pires durées d'exécution des tâches ne conduisent pas forcément à leurs pires temps de réponse comme l'illustre l'exemple présenté sur la figure 2.1. Cette figure présente les chronogrammes de l'exécution d'un système de quatre tâches $(\tau_i = (T_i, C_i, P_i))_{i\in[\![1,4]\!]}$ sans gigue ni *offset* et à échéances sur requêtes $(D_i = T_i)$. Les tâches sont ordonnancées selon une politique à priorités fixes, la priorité de τ_i étant notée P_i. Nous avons $\tau_1 = (18, 5, 4)$, $\tau_2 = (18, 1, 1)$, $\tau_3 = (18, C_3, 2)$, avec $C_3 \in \{3, 5\}$ et $\tau_4 = (6, 2, 3)$. Enfin, il y a un lien de précédence entre τ_2 et τ_1 : la tâche τ_1 ne peut être exécutée qu'après une exécution de τ_2. Les requêtes d'activations sont représentées par des flèches vers le haut et la fin des tâches par des flèches vers le bas.

Nous constatons que lorsque $C_3 = 5$, le système est ordonnançable. Par contre, lorsque $C_3 = 3$, la tâche τ_2 peut être exécutée plus tôt, libérant la tâche la plus prioritaire et la plus longue τ_1. Par conséquent, τ_4 finit par violer son échéance.

Des techniques d'évaluation plus fine des temps de blocage dûs à des précédences sont également étudiées, par exemple dans [HKL91, PH99].

Dans le cas de l'utilisation d'un protocole d'accès aux ressources comme *Priority Ceiling Protocol* [SRL90], la méthode classique consiste à évaluer des *facteurs de blocage*, temps maximums pendant lesquels les tâches peuvent être bloquées en attente des ressources partagées, qui s'ajoutent aux temps de réponse.

L'approche analytique fournit donc des algorithmes de complexité faible (polynomiale) permettant de décider l'ordonnançabilité d'un ensemble de tâches, de façon exacte si les tâches sont indépendantes ou de façon généralement pessimiste si les tâches se synchronisent entre elles.

2.3 Approches formelles

Le besoin de résultats plus précis et surtout de modèles plus complexes plaide donc en faveur de l'utilisation des modèles dits « formels ». Ils permettent notamment la prise en compte de mécanismes de synchronisation complexes et de durées d'exécution des tâches variables.

2.3.1 Modèles temporisés

Les modèles formels temporisés tels que les automates temporisés et les réseaux de Petri T-temporels permettent de modéliser l'évolution d'un système suite à des actions discrètes ou à l'écoulement continu du temps. Le temps est représenté de façon dense ce qui implique que ces modèles ont en général un espace d'états infini.

L'approche *symbolique* consiste à partitionner l'espace d'états en un nombre fini de groupes d'états partageant une ou plusieurs propriétés « intéressantes », par exemple même marquage pour un réseau de Petri T-temporel ou accessibilité par la même séquence de transitions discrètes. Cela peut se traduire en : soit une relation d'équivalence $\mathcal{R}$ sur les états du

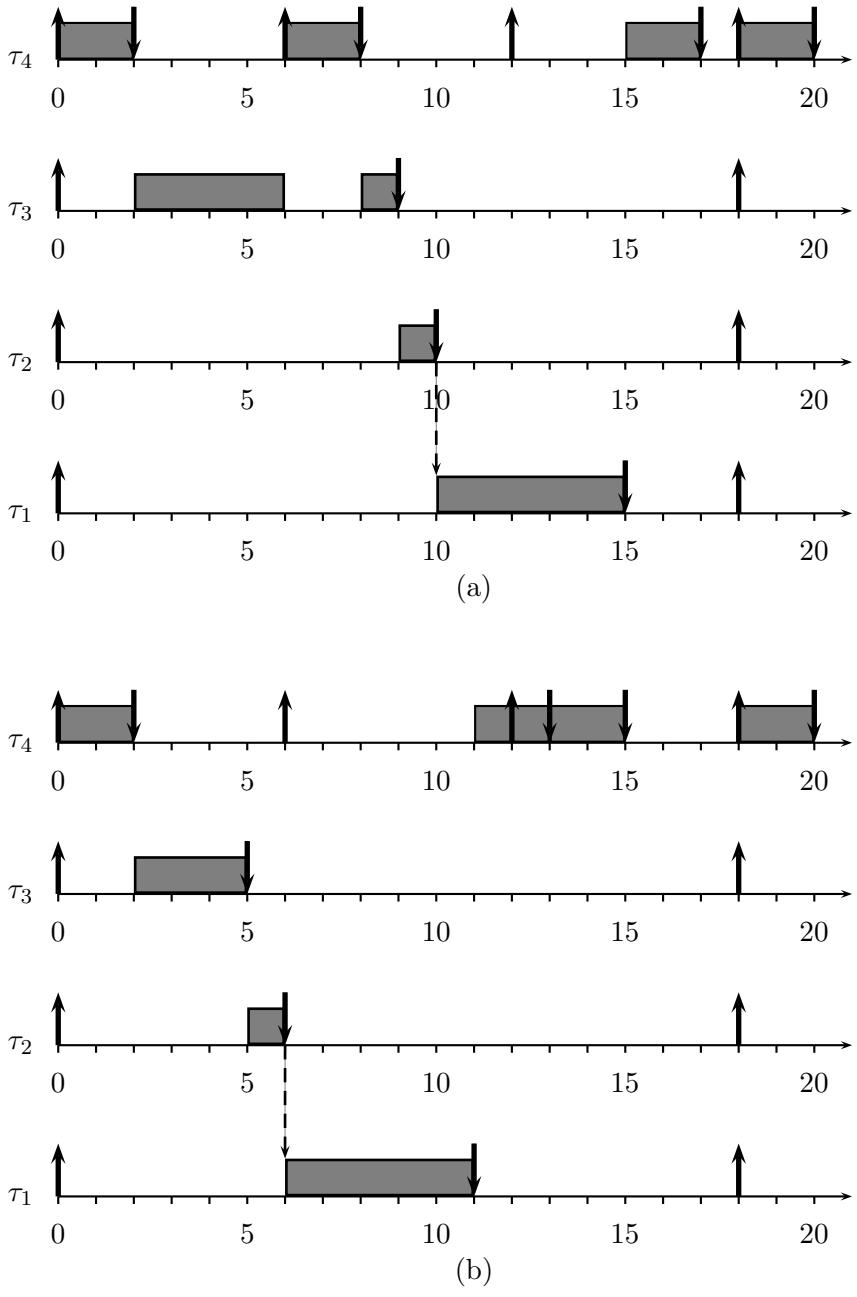

FIG. 2.1 – Chronogramme obtenu avec (a) $C_3 = 5$ et (b) $C_3 = 3$

système telle que deux états en relation par $\mathcal{R}$ partagent les propriétés P_i, calculer les classes d'équivalence définies par $\mathcal{R}$.

Automates temporisés

Les automates temporisés [AD94] étendent les automates finis classiques avec des horloges explicites. Dans ce modèle, nous pouvons soit rester dans la même localité et laisser écouler le temps, soit prendre une transition discrète et effectuer l'action associée et éventuellement remettre certaines horloges à 0. Pour chaque transition est définie une *garde* qui contraint les horloges de l'automate. Cette garde doit être vérifiée pour que la transition puisse être franchie. Henzinger *et al.* étendent ce modèle à l'aide de contraintes sur les horloges, appelées *invariants* associés aux localités [HNSY94]. Le séjour dans une localité n'est possible que si l'invariant associé est vérifié.

La première méthode de calcul de l'espace d'états consiste à le partitionner en *régions* [AD94]. Les régions regroupent les états de l'automate temporisés selon une partition « géométrique » de l'espace des horloges. Le nombre de régions est fini mais très grand ce qui rend l'utilisation de cette partition inutilisable pour la vérification mais cela prouve la décidabilité du problème de l'accessibilité d'états pour les automates temporisés.

La méthode la plus utilisée consiste à regrouper les régions en *zones* [LPY95] représentées par des *Difference Bounds Matrices* (DBM)[3]. Les états à l'intérieur d'une zone sont tous les états accessibles entre le franchissement de deux transitions de l'automate.

Deux outils principaux de vérification sont disponibles pour les automates temporisés : Uppaal [LPY97] et Kronos [Yov97].

Réseaux de Petri et le temps

Les deux extensions temporelles principales des réseaux de Petri sont les réseaux de Petri *temporisés* [Ram74] et les réseaux de Petri *temporels* [Mer74]. Pour les premiers, le temps est représenté par des durées minimales (ou exactes dans le cas d'un fonctionnement « au plus tôt » du réseau) de franchissement des transitions. Pour les seconds, le temps prend la forme d'un intervalle contraignant les instants de tir des transitions.

Par ailleurs le temps peut être associé aux transitions, aux places, aux jetons, aux arcs... Les classes de réseaux de Petri temporisés qui en résultent sont incluses dans les classes de réseaux de Petri temporels correspondantes [PT99]. Les principales classes de réseaux de Petri temporels sont les réseaux de Petri T-temporels [BD91], P-temporels [KDC96] et A-temporels [dFRA00, AN01], où un intervalle de temps est associé respectivement aux transitions, aux places et aux arcs. Les réseaux de Petri temporels à flux (*Time Stream Petri Nets*) [DS94] sont également une extension temporelle des réseaux de Petri où un intervalle de temps est associé aux arcs et ont été introduits pour modéliser les applications multimédia.

La méthode classique d'analyse des réseaux de Petri T-temporels est le *graphe des classes d'états*[4] [BM83, BD91]. Dans cette méthode, l'espace d'états est partitionné selon la relation d'équivalence suivante : deux états sont en relation si et seulement si ils sont accessibles par une même séquence de transitions discrètes. En particulier, ils partagent le même marquage. Le graphe des classes d'états préserve les marquages et le langage discret du graphe des états accessibles. Il est donc adapté à la vérification de l'accessibilité de marquage et de propriétés

[3] Voir 3.2
[4] Voir 4

LTL. Par contre, il ne préserve pas la structure de branchement du graphe des états ce qui rend impossible la vérification de propriétés CTL*.

Pour remédier à ce problème, Yoneda et Ryuba proposent dans [YR98] un autre partitionnement de l'espace d'états en *classes atomiques*. Dans une classe atomique, tout état a un successeur dans chacune des classes atomiques obtenues par tir d'une transition. Les classes atomiques de Yoneda et Ryuba sont obtenues en découpant les classes d'états (définies de façon différente de [BD91]), par l'ajout de contraintes linéaires. Les auteurs prouvent que le graphe des classes atomiques permet la vérification de propriétés CTL*. Cependant cette technique n'est pas applicable aux réseaux pour lesquels les intervalles de temps ne sont pas bornés.

Lilius a proposé une amélioration de cette technique [Lil99] permettant l'utilisation de techniques d'ordre partiel développées pour les systèmes non temporisés.

Dans [BV03], Berthomieu et Vernadat proposent une construction alternative des classes d'états atomiques, qui fournit un graphe plus compact, se calcule plus rapidement et est applicable aux réseaux dont les intervalles de temps ne sont pas forcément bornés.

Par ailleurs, les problèmes « intéressants » pour la vérification, tels que l'accessibilité de marquages ou d'états sont décidables pour les réseaux de Petri T-temporels bornés[5].

Traduction des réseaux de Petri temporels en automates temporisés

Dans [SY96], Sifakis et Yovine étudient une sous-classe des réseaux de Petri temporels à flux dont le réseau sous-jacent est sauf (STPN). Dans ce modèle, étant données une transition t et une place amont p de t, un intervalle de temps $[l, u]$ est associé à l'arc (p, t). Un jeton arrivant dans la place p doit attendre une durée comprise entre l et u avant d'être disponible pour la transition t. Les auteurs proposent une traduction de ces réseaux en automates temporisés : une horloge est associée à chaque *place* du réseau. Elle est remise à zéro à chaque fois que la place reçoit un jeton. Les localités de l'automate correspondent au graphe des marquages du réseau de Petri. Les gardes et les invariants sont dérivés des intervalles associés aux arcs.

Dans [BST98], Bornot, Sifakis et Tripakis s'intéressent aux réseaux de Petri avec échéances (PND) qui sont des réseaux de Petri saufs, étendus avec des horloges. Un PND peut être défini comme un automate temporisé à échéances (TAD) dont la structure discrète est le graphe des marquages du réseau de Petri. Les transitions de ce TAD sont soumises aux même contraintes temporelles que les transitions du PND. Le PND et le TAD ont le même nombre d'horloges. Les auteurs proposent une traduction des réseaux de Petri temporels en TAD avec une horloge par arc entrant du réseau de Petri temporel initial. Les automates temporisés à échéances pouvant être considérés comme des automates temporisés standards, la méthode fournit donc, par transitivité, une traduction des réseaux de Petri temporels dont le réseau sous-jacent est sauf vers les automates temporisés. Le résultat possède un nombre d'horloges supérieur ou égal au nombre de transitions du réseau de Petri initial.

Gardey et Roux proposent une méthode pour appliquer le calcul du graphe des régions des automates temporisés aux réseaux de Petri T-temporels bornés, en utilisant des DBM [GRR03, GRR05]. Ce calcul peut être utilisé pour générer un automate temporisé des marquages du réseau de Petri qui lui est temporellement bisimilaire en s'inspirant de la méthode de Sava et Alla [Sav01]. Cet automate possède une localité par marquage et une horloge par transition. La vérification sur cet automate n'est donc pas très efficace car sa complexité

[5] Voir 4.2.1

dépend exponentiellement du nombre d'horloges. Les auteurs montrent que l'application des algorithmes de réduction du nombre d'horloges de Daws et Yovine [DY96] permet de réduire sensiblement le nombre d'horloges du résultat.

Cassez et Roux ont développé une méthode de traduction structurelle d'un réseau de Petri T-temporel en un automate temporisé [CR03, CR04]. Chaque transition du réseau de Petri est encodée dans un automate temporisé avec variables. Tous les automates ainsi obtenus sont combinés avec un superviseur en un produit synchronisé d'automates temporisés. Le calcul est très rapide et applicable à tous les réseaux de Petri T-temporel même non bornés car il est structurel. La méthode n'est cependant pas bien adaptée à la vérification car le produit d'automates possède une horloge par transition du réseau de Petri T-temporel et ce nombre ne peut être réduit par des algorithmes comme celui de Daws et Yovine [DY96] à cause de la structure de produit du résultat.

2.3.2 Modèles pour l'ordonnancement

Les modèles temporisés ne sont en général pas suffisants pour modéliser et vérifier les applications temps réel. En effet, dans ces modèles, le temps s'écoule de façon identique pour toutes les composantes du système, ce qui permet de modéliser des politiques d'ordonnancement non préemptives ou des tâches qui s'exécutent chacune sur un processeur différent.

Par contre, ils ne permettent pas de représenter les politiques d'ordonnancement préemptives où l'exécution d'une tâche est interrompue et reprise au même endroit un peu plus tard.

Par ailleurs, nous pouvons remarquer que le cas où des tâches partagent un même processeur n'est pas forcément « pire » que le cas où chaque tâche a son propre processeur comme le montre l'exemple de la figure 2.2. Cette figure présente les chronogrammes d'un système de deux tâches τ_1 et τ_2 partageant une ressource critique. La période de τ_1 est 4 et sa durée d'exécution est 2, la deuxième moitié étant en section critique (représentée en gris foncé). La période de τ_2 est 8 et sa durée d'exécution est 4 entièrement en section critique.

Dans le premier scénario (figure 2.2a), les tâches partagent un même processeur selon une politique d'ordonnancement à priorités fixes. τ_1 est la tâche la plus prioritaire. Dans ce cas, le système est ordonnançable.

La figure 2.2b présente un scénario pour leqeul chaque tâche est exécutée sur un processeur différent. Dans ce cas, τ_2 prend la ressource critique en premier et empêche τ_1 de respecter son échéance. Le système n'est donc pas ordonnançable.

Par conséquent, une propriété vraie pour un modèle temporisé n'est pas forcément vraie pour un modèle plus précis prenant en compte l'ordonnancement. Il est donc nécessaire d'utiliser des modèles plus expressifs permettant de modéliser l'interruption d'une action et sa reprise au même endroit. Cependant, ce gain en expressivité ce traduit par l'indécidabilité de la plupart des problèmes liés à la vérification et par une complexité accrue des semi-algorithmes mis en œuvre pour les résoudre.

Automates et ordonnancement

Les automates hybrides étendent les automates à états finis classiques à l'aide de variables continues dont les dynamiques d'évolution sont spécifiées dans chaque localité à l'aide d'une équation différentielle.

Les automates hybrides linéaires forment une sous-classe des automates hybrides. Dans ce modèle, l'équation différentielle régissant l'évolution du vecteur de variables continues X est

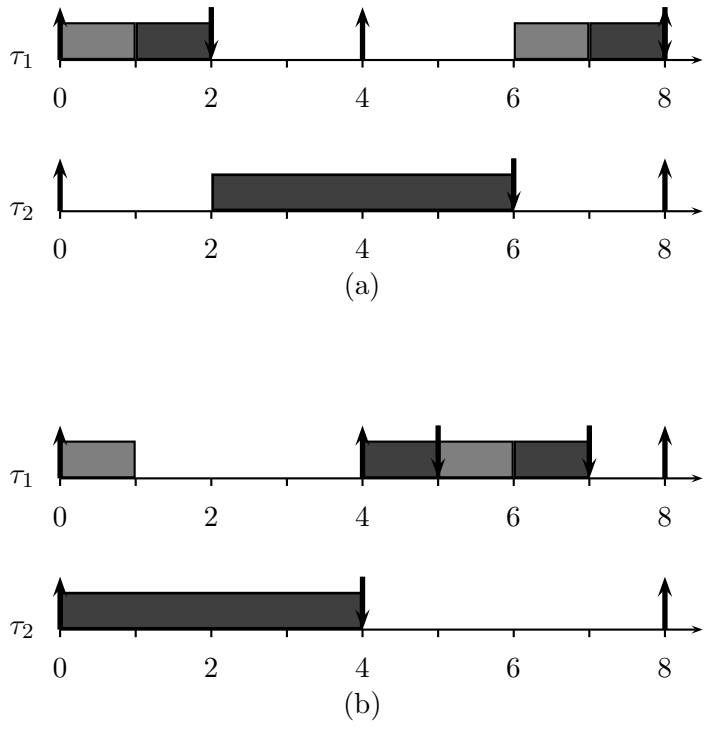

FIG. 2.2 – Chronogramme obtenu avec (a) un processeur et (b) deux processeurs

de la forme $A\dot{X} \leq B$, où A est une matrice réelle et B un vecteur réel. Pour cette sous-classe, des algorithmes de vérification symboliques ont été développés [AHH96] et implémentés dans l'outil HyTech[HHWT97].

Les automates à chronomètres [CL00] peuvent être définis comme des automates temporisés pour lesquels les horloges peuvent être arrêtées et redémarrées plus tard avec la même valeur. Ces horloges sont appelées chronomètres (*stopwatches*). Il s'agit donc d'une sous-classe *syntaxique* des automates hybrides linéaires.

Cassez et Larsen montrent dans [CL00] que les automates à chronomètres avec des délais inobservables sont aussi expressifs que les automates hybrides linéaires [ACH+95], dans le sens où, pour tout langage acceptable par un automate hybride linéaire, il existe un automate à chronomètres qui accepte le même langage. Le problème de l'accessibilité étant indécidable pour les automates hybrides linéaires, ils en déduisent qu'il l'est également pour les automates à chronomètres. Les auteurs proposent également une surapproximation de l'espace atteignable à l'aide de DBM.

Dans [MV94], McManis et Varaiya proposent une extension des automates temporisés, appelée automates à suspension, où les variables continues progressent de façon similaire à [CL00]. Ils prouvent que, pour ce modèle, l'accessibilité est indécidable et se ramènent à un cas décidable, similaire à celui des automates temporisés, en considérant des durées de suspension fixes et entières. Lorsqu'un chronomètre est arrêté puis redémarré, la durée de la suspension est retirée à sa valeur. Dans cette approche, les relations de précédence induisant les préemptions doivent être encodées explicitement en termes de localités de l'automate ce qui peut limiter l'expressivité et la facilité d'utilisation du modèle. Par ailleurs comme nous l'avons vu dans la section 2.2, le cas de durées d'exécution fixes ne permet pas de modéliser certaines anomalies où une tâche se terminant plus tôt conduit à un temps de réponse plus long d'une autre tâche.

L'approche de McManis et Varaiya est développée par Fersman, Mokrushin, Petterson et Yi [FPY02, FMPY03]. En s'appuyant sur des automates avec tâches qui modélisent les arrivées des tâches, les auteurs proposent une analyse de l'ordonnançabilité du système sous la forme d'un problème d'accessibilité dans l'espace d'états d'automates à soustractions représentant l'ordonnanceur. Ils prouvent par ailleurs que ce problème est décidable sous les contraintes suivantes : les tâches sont indépendantes et les durées d'exécution varient dans des intervalles [FPY02] ou les tâches communiquent entre elles mais les durées d'exécution sont fixes et entières [FMPY03]. Cela n'est pas étonnant puisque dans le cas de tâches indépendantes, nous avons vu que le pire cas est obtenu quand la durée d'exécution des tâches est maximale.

Une approche intéressante, proposée par Altisen *et al.* [AlP+99, AlS00, AGS02], s'appuie sur le paradigme de la synthèse de contrôleur. En effet, l'ordonnanceur d'un système temps réel peut être vu comme un contrôleur du système composé des différentes tâches. L'idée est donc d'obtenir, par construction, l'ordonnançabilité du système modélisé en temps discret en restreignant les gardes des événements contrôlables. Ces restrictions se font par des invariants de contrôle impliquant des contraintes modélisant l'ordonnançabilité et la politique d'ordonnancement. Cependant, cette approche peut s'avérer difficile à mettre en œuvre car la recherche des invariants de contrôle peut être ardue.

Réseaux de Petri T-temporels et ordonnancement

Un certain nombre d'auteurs proposent d'étendre les réseaux de Petri T-temporels afin de prendre en compte les aspects liés à l'interruption et à la reprise d'actions.

Okawa et Yoneda [OY96] proposent une extension dans laquelle les transitions sont groupées et où chaque groupe possède une vitesse d'exécution. Les groupes de transitions correspondent à des transitions modélisant des activités concurrentes et qui sont prêtes à être tirées simultanément. Dans ce cas, leur vitesse d'exécution est divisée par la somme des vitesses d'exécution : ils se placent donc dans le cas *fluide*.

Roux et Déplanche [RD02] ont étendu les réseaux de Petri T-temporels pour prendre en compte la façon dont les tâches réparties sur les différents processeurs sont ordonnancées. Ils proposent un calcul de l'espace d'états basé sur le graphe de classes d'états classique et les DBM. Nous montrerons dans cette thèse que ce calcul est en fait une surapproximation.

La même approche est développée dans [BFSV04, BFSV03], avec les réseaux de Petri préemptifs qui associent aux transitions des ressources et des priorités d'accès à ces ressources. Dans [BFSV04], les auteurs proposent en outre une méthode intéressante permettant une analyse temporelle quantitative du réseau. Étant donnée une séquence de transitions non temporisée issue du graphe des classes d'états surapproximé, ils retrouvent sous la forme d'un problème de programmation linéaire, les durées possibles séparant les tirs de transitions de la séquence. Ils peuvent ainsi vérifier si cette séquence est effectivement possible pour le réseau ou si elle a été introduite par la surapproximation. Dans le cas d'une séquence effectivement faisable, ils peuvent en déduire des bornes sur les durées séparant des tirs de transitions et donc des propriétés temporelles de l'application modélisée.

Enfin un modèle plus général de réseaux de Petri mettant en œuvre des chronomètres a été proposé dans [RL04] fusionnant les deux modèles classiques des réseaux de Petri T-temporels et des réseaux de Petri à hyperarcs inhibiteurs. Pour ces réseaux, le problème du calcul de l'espace d'états est le même que pour les réseaux de Petri temporels étendus de [RD02] et [BFSV04] et pour ces trois modèles, comme pour les réseaux de Petri T-temporels, la bornitude et l'accessibilité sont indécidables. Ils sont donc obtenus en explorant l'espace d'états à l'aide de semi-algorithmes.

2.4 Notre contribution

Dans cette thèse, nous nous intéressons aux réseaux de Petri T-temporels comme formalisme de modélisation des systèmes temps réel. La méthode classique de vérification sur ce modèle étant le calcul du graphe des classes d'états, nous proposons des améliorations du calcul des classes d'états permettant d'améliorer sa complexité temporelle ou spatiale.

Par ailleurs, le graphe des classes d'états étant peu adapté à la vérification de propriétés temporelles, nous proposons une construction originale permettant d'obtenir un automate temporisé représentant l'espace d'états du réseau plutôt qu'un graphe. Cet algorithme de traduction est doté d'une condition de convergence moins restrictive permettant d'obtenir un résultat plus compact que le graphe des classes d'états et d'un mécanisme de réduction du nombre d'horloges à la volée. Nous proposons également un cadre formel pour exprimer des propriétés TCTL[6] directement sur le réseau de Petri T-temporel et les vérifier ensuite par l'intermédiaire de sa traduction en automate temporisé. Cette vérification peut être réalisée à

[6]Timed Computation Tree Logic.

l'aide d'outils comme UPPAAL ou KRONOS de façon efficace grâce au nombre réduit d'horloges de l'automate produit.

Par ailleurs, afin de modéliser plus finement les systèmes temps réel, nous nous intéressons aux réseaux de Petri T-temporels étendus à l'ordonnancement de [RD02]. Nous prouvons que la plupart des problèmes liés à la vérification sont indécidables pour ce modèle, même borné, et nous nous intéressons à l'élaboration de semi-algorithmes de calcul de l'espace d'états. Nous montrons d'abord comment adapter le calcul du graphe des classes à ce modèle avec d'une part, l'utilisation de polyèdres généraux pour un calcul exact et d'autre part, une approximation de ces polyèdres à base de *Difference Bounds Matrices* pour un calcul plus rapide mais surapproximant. Ensuite nous adaptons la traduction en automate temporisé développée pour les réseaux de Petri T-temporels afin d'obtenir un automate à chronomètres dans le cas de l'extension à l'ordonnancement. Comme dans le cas temporisé, l'algorithme dispose d'une réduction du nombre de chronomètres à la volée ce qui rend la vérification à l'aide d'outils comme HYTECH plus efficace. Enfin nous montrons que nous pouvons utiliser le calcul surapproximant pour la traduction et obtenir tout de même un automate à chronomètres temporellement bisimilaire au réseau de Petri. Par conséquent, la phase de traduction est efficace et rapide comparée au temps requis pour la vérification en elle-même.

Enfin, nous montrons comment prendre en compte le cas de tâches concurrentes s'exécutant avec la même priorité sur un même processeur en modélisant l'algorithme classique du tourniquet ou *Round-Robin* et comment modéliser et vérifier des systèmes ordonnancés par la politique à priorités dynamique *Earliest Deadline First*.

2.5 Plan de la thèse

Ces deux premiers chapitres font partie de la première partie de ce document. La suite est organisée de la façon suivante :

Le chapitre 3 présente les notations générales ainsi que des notions préliminaires utilisées dans tout ce document.

La deuxième partie concerne la classe des modèles temporisés. Elle présente les réseaux de Petri T-temporels, le calcul du graphe des classes d'états, leur traduction en automates temporisés et la vérification en pratique de propriétés TCTL.

Le chapitre 4 donne la définition formelle des réseaux de Petri T-temporels ainsi que la méthode de calcul du graphe des classes d'états.

Le chapitre 5 décrit la traduction par le calcul de l'espace d'états des réseaux de Petri T-temporels bornés en automates temporisés. Nous donnons également dans ce chapitre un cadre formel d'expression et de vérification de propriétés TCTL pour les réseaux de Petri T-temporels bornés.

Dans la troisième partie, nous nous plaçons dans le cadre des modèles à chronomètres. Nous étudions les réseaux de Petri T-temporels étendus à l'ordonnancement, du point de vue de leur expressivité et de la décidabilité des problèmes qui leurs sont liés. Nous adaptons également les algorithmes de la partie précédente à ce nouveau modèle. Finalement nous montrons comment étendre le modèle pour la prise en compte d'autres politiques d'ordonnancement.

Le chapitre 6 donne la définition formelle des réseaux de Petri T-temporels étendus à l'ordonnancement. Nous y étudions l'expressivité de ce modèle et la décidabilité des principaux problèmes liés à la vérification. Enfin nous y adaptons le calcul du graphe des classes d'états.

Dans le chapitre 7, nous proposons d'adapter la traduction du chapitre 5 pour l'extension

à l'ordonnancement. Cela nous permet d'obtenir un automate à chronomètres temporellement bisimilaire au réseau de Petri traduit.

Enfin avec le chapitre 8, nous étendons le modèle des réseaux de Petri T-temporels étendus à l'ordonnancement pour la prise en compte des politiques d'ordonnancement *Round-Robin* et *Earliest Deadline First*.

Dans la quatrième partie, nous réalisons une étude de cas.

En préliminaire, dans le chapitre 9, nous décrivons l'implémentation des différents semi-algorithmes développés dans cette thèse. Nous proposons également deux améliorations du calcul des classes d'états permettant de diminuer sa complexité temporelle ou spatiale.

Puis nous réalisons l'étude de cas proprement dite dans le chapitre 10.

Finalement nous concluons et donnons quelques perspectives de développement de nos travaux dans la dernière partie.

Chapitre 3

Préliminaires

Dans ce chapitre, nous présentons les notations utilisées tout au long de ce document ainsi que des notions de base sur les systèmes de transitions et les polyèdres.

3.1 Notations générales

- $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ désignent respectivement l'ensemble des entiers naturels, des rationnels et des réels ;
- $\mathbb{R}^+$ est l'ensemble des réels positifs ou nuls. $\mathbb{R}_*^+$ est l'ensemble des réels strictement positifs ;
- soit $n \in \mathbb{N}$, $\mathbb{R}^n$ désigne l'espace réel à n dimensions ;
- soit $(m, n) \in \mathbb{N}^2$, $\mathcal{M}_{m,n}(\mathbb{R})$ est l'ensemble des matrices réelles à m lignes et n colonnes. $\mathcal{M}_n(\mathbb{R})$ est l'ensemble des matrices réelles carrées de dimension n ;
- étant donnés deux entiers naturels m et n, $[\![m, n]\!]$ désigne l'ensemble des entiers compris entre m et n, incluant m et n ;
- soit un ensemble fini E. Nous notons $|E|$ le cardinal de E ;
- $\wedge$ désigne l'opérateur logique `et`, $\vee$ l'opérateur logique `ou` et $\neg$ l'opérateur logique `non` ;
- soit V un ensemble de variables. Nous notons $\mathcal{C}(V)$ l'ensemble des *contraintes entières simples* sur V, c'est-à-dire l'ensemble des combinaisons booléennes (avec les opérateurs logiques $\vee$, $\wedge$ et $\neg$) de termes de la forme $v - v' \sim c$ ou $v \sim c$, avec $v, v' \in V$, $\sim \in \{<, \leq, =, \geq, >\}$ et $c \in \mathbb{N}$.

3.1.1 Systèmes de transitions temporisés

En suivant l'idée de [LPY95], nous définissons un système de transitions temporisé (*Timed Transition System* en anglais, ou TTS) comme un système de transitions pour lequel deux types de transitions sont possibles : des transitions d'action et des transitions de temps modélisant respectivement des évolutions *discrètes* et des évolutions *continues* du système.

Définition 2 (Système de transitions temporisé) *Soit A un ensemble d'actions (ou* alphabet*). Pour $d \in \mathbb{R}^+$, nous notons $\epsilon(d)$ l'action de A ($\epsilon(d) \in A$) consistant à laisser passer le temps d. Un système de transitions temporisé sur A est un triplet $(S, s_0, \rightarrow)$ où :*

- *S est un ensemble d'*états *;*
- *$s_0 \in S$ est l'état initial ;*

– $\rightarrow$ *est la relation de transition qui se décompose en une relation de transition* continue $\overset{\epsilon(d),d\in\mathbb{R}^+}{\rightarrow}$ *et une relation de transition* discrète $\overset{a\in A}{\rightarrow}$.

3.1.2 Quotient

Soit $\mathcal{S} = (S, s_0, \rightarrow)$ un système de transitions étiqueté sur un alphabet A. Soit $\mathcal{R} \in S \times S$ une relation d'équivalence sur S.

Le quotient de $\mathcal{S}$ par $\mathcal{R}$, noté $\mathcal{S}/\mathcal{R}$, est un système de transitions $(Q, q_0, \rightsquigarrow)$ étiqueté sur A, défini par :

– Q est l'ensemble des classes d'équivalence de $\mathcal{R}$;
– q_0 est la classe d'équivalence de s_0 : $s_0 \in q_0$;
– $\rightsquigarrow$ est définie par : soit $q, q' \in Q \times Q$ et $a \in A$, $q \overset{a}{\rightsquigarrow} q' \Leftrightarrow \exists (s, s') \in S \times S$ t.q. $s \in q, s' \in q'$ et $s \overset{a}{\rightarrow} s'$.

3.1.3 Langage

Définition 3 (Séquence de transitions) *Soit un système de transitions $\mathcal{S} = (S, s_0, \rightarrow),$, étiqueté sur un alphabet A. $(a_1, a_2, \ldots, a_n), n \in \mathbb{N}_*$ est une* séquence de transitions *du système $\mathcal{S}$ si $\exists (s_1, \ldots, s_{n+1}) \in S^{n+1}$ t.q. $s_1 \overset{a_1}{\rightarrow} s_2 \overset{a_2}{\rightarrow} \cdots \overset{a_n}{\rightarrow} s_{n+1}$. Si $s_1 = s_0$ alors $(a_1, a_2, \ldots, a_n)$ est un* mot *de $\mathcal{S}$*

Le *langage* d'un système de transition étiqueté est l'ensemble de ses mots. Pour un système de transitions temporisé, nous distinguerons le *langage temporisé* constitués de mots qui contiennent à la fois des actions discrètes et continues et le *langage non temporisé* qui est constitué des mots dans lesquels les actions continues ont été abstraites.

3.1.4 Bisimulation

Soient deux systèmes de transitions $\mathcal{S}_1$ et $\mathcal{S}_2$. Nous pouvons définir une équivalence de comportement entre $\mathcal{S}_1$ et $\mathcal{S}_2$ par l'intermédiaire d'une relation de bisimulation. Celle-ci assure que toute action de l'un des deux systèmes peut être *simulée* par l'autre, c'est-à-dire qu'elle a un équivalent dans cet autre système.

Définition 4 (Bisimulation) *Soient $\mathcal{S}_1 = (S_1, s_1^0, \rightarrow_1)$ et $\mathcal{S}_2 = (S_2, s_2^0, \rightarrow_2)$ deux systèmes de transitions étiquetés sur un même ensemble d'étiquettes A. Soit $\mathcal{R} \in S_1 \times S_2$ une relation binaire sur l'ensemble des états des deux systèmes.*

$\mathcal{R}$ est une bisimulation ssi $\forall (s_1, s_2) \in S_1 \times S_2$ t.q. $s_1 \mathcal{R} s_2, \forall a \in A,$

$$\begin{cases} \exists s_1' \in S_1 \text{ t.q. } s_1 \overset{a}{\rightarrow}_1 s_1' \Rightarrow \exists s_2' \in S_2 \text{ t.q. } s_2 \overset{a}{\rightarrow}_2 s_2' \text{ et } s_1' \mathcal{R} s_2', \\ \exists s_2' \in S_2 \text{ t.q. } s_2 \overset{a}{\rightarrow}_2 s_2' \Rightarrow \exists s_1' \in S_1 \text{ t.q. } s_1 \overset{a}{\rightarrow}_1 s_1' \text{ et } s_1' \mathcal{R} s_2' \end{cases}$$

Si $\mathcal{S}_1$ et $\mathcal{S}_2$ sont des systèmes de transitions temporisés, nous dirons que $\mathcal{R}$ est une bisimulation *temporelle* si c'est une bisimulation à la fois pour la relation de transition continue et pour la relation de transition discrète.

Par ailleurs, nous pouvons remarquer que deux systèmes de transitions bisimilaires ont même langage. Et, par conséquent, deux TTS temporellement bisimilaires ont même langage temporisé.

3.2 Polyèdres

Un *polyèdre* est une région de l'espace réel à n dimensions définie par un système d'inéquations linéaires en X, $A.X \leq B$, avec $A \in \mathcal{M}_{m,n}(\mathbb{R})$ une matrice à m lignes et n colonnes et $B \in \mathbb{R}^m$ un vecteur donnés, et $X \in \mathbb{R}^n$ le vecteur inconnu.

Avec cette définition, nous voyons que certains points peuvent être rejetés à l'infini. Nous appellerons *polytope* un polyèdre *borné*.

Motzkin a montré que tout polyèdre peut également être représenté comme l'union d'un polytope et d'un cône polyédrique (voir par exemple [Sch86]). Chacune des deux représentations admet des formes minimales et une *forme canonique* unique à un choix de base de sous-espace vectoriel près [AFP02].

3.2.1 Difference Bounds Matrix

Dans le cas particulier où chaque ligne de A contient exactement un 1, au plus un -1 et pour le reste que des 0, le système $A.X \leq B$ peut-être représenté sous la forme d'une matrice unique appelée *matrice de bornes de différences* ou *Difference Bounds Matrix* (DBM) en anglais [BM83, Dil89].

En effet, les inéquations du système sont alors de la forme « $x_i - x_j \leq b_k$ » et « $x_i \leq b_k$ » avec $i, j \in [\![1, n]\!]$ et $k \in [\![1, m]\!]$. Soit x_0 une variable telle que $x_0 = 0$, alors nous pouvons écrire toutes les inéquations sous la forme $x_i - x_j \leq b_k$, avec $i, j \in [\![0, n]\!]$. Le système peut donc être représenté par la matrice $D \in \mathcal{M}_{n+1}(\mathbb{R})$ telle que si d_{ij} est l'élément de $\mathcal{D}$ situé sur la i-ème ligne et la j-ième colonne [1], $d_{ij} = b_k$. $\mathcal{D}$ est une DBM.

Dans le cas où certaines contraintes sont strictes, les éléments de la DBM sont des couples $(b_k, \sim)$ avec $\sim \in \{<, \leq\}$.

3.2.2 Complexité

Soit n la dimension de l'espace, c'est-à-dire le nombre de variables du système, et m le nombre de sommet du polyèdre. Le calcul de la forme canonique a une complexité au pire cas polynomiale en n et en m [AFP02].

Pour les DBM, l'inclusion, l'intersection, l'égalité ont des complexités quadratiques en n ($O(n^2)$). Le test du vide et la mise sous forme canonique se font en $O(n^3)$.

[1] Si pour un même couple (i, j), il existe plusieurs b_k nous choisissons le plus petit.

Deuxième partie

Modèles temporisés

Chapitre 4

Réseaux de Petri T-temporels

Dans ce chapitre nous introduisons les réseaux de Petri T-temporels . Nous en faisons d'abord une présentation informelle avant de donner les définitions de la syntaxe et de la sémantique du modèle. Enfin nous nous intéressons au calcul de l'espace d'états des réseaux de Petri T-temporels par la méthode du graphe des classes d'états.

4.1 Présentation informelle

Les réseaux de Petri T-temporels, ou *time Petri nets* (TPN) en anglais, sont une extension des réseaux de Petri introduite par Merlin en 1974 [Mer74]. Ils consistent à ajouter du temps au modèle classique sous la forme d'un intervalle $[\alpha(t), \beta(t)]$ de $\mathbb{R}$ associé à chaque transition t du réseau [1] . Pour être tirée, une transition t doit non seulement être sensibilisée mais également l'avoir été continûment pendant une durée comprise entre $\alpha(t)$ et $\beta(t)$.

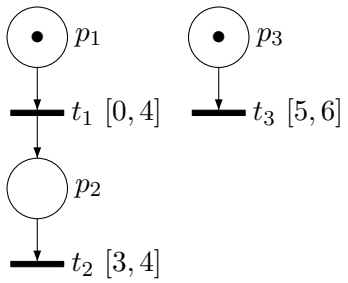

FIG. 4.1 – Un réseau de Petri T-temporel

La figure 4.1, issue de [Gal97], montre un exemple de réseau de Petri T-temporel. Les places p_1 et p_3 sont marquées donc les transitions t_1 et t_3 sont *sensibilisées*, car elles possèdent un jeton dans leurs places amont. Cependant, seule t_1 est *tirable* car les intervalles de tir sont disjoints. Le tir de t_1 amène à un nouvel état dans lequel p_2 et p_3 sont marquées.

Nous allons maintenant définir plus formellement les réseaux de Petri T-temporels.

[1] $\beta(t)$ peut également prendre pour valeur $+\infty$.

4.2 Définitions

Définition 5 (Réseau de Petri T-temporel) *Un réseau de Petri T-temporel est un 7-uplet* $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0)$*, où*

- $P = \{p_1, p_2, \ldots, p_m\}$ *est un ensemble fini et non vide de* places *;*
- $T = \{t_1, t_2, \ldots, t_n\}$ *est un ensemble fini et non vide de* transitions $(T \cap P = \emptyset)$ *;*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *est la* fonction d'incidence amont *;*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *est la* fonction d'incidence aval *;*
- $M_0 \in \mathbb{N}^P$ *est le* marquage initial *du réseau ;*
- $\alpha \in (\mathbb{R}^+)^T$ *et* $\beta \in (\mathbb{R}^+ \cup \{\infty\})^T$ *sont les fonctions donnant pour chaque transition, respectivement son instant de tir* au plus tôt *et* au plus tard *$(\alpha \leq \beta)$.*

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.

Une transition t est dite *sensibilisée* par le marquage M si $M \geq {}^\bullet t$, c'est-à-dire si le nombre de jetons, pour M, de chaque place amont de t est plus grand ou égal à la valuation de l'arc entre cette place et la transition. Nous notons $t \in enabled(M)$.

Soit un marquage M du réseau et une transition t'. Supposons que nous pouvons tirer t'. Alors, une transition t est dite *nouvellement sensibilisée* par le tir de la transition t' à partir du marquage M, ce que nous noterons $\uparrow enabled(t, M, t')$, si t est sensibilisée par le nouveau marquage $M - {}^\bullet t' + t'^\bullet$ mais ne l'était pas par le marquage $M - {}^\bullet t'$. Formellement,

$$\uparrow enabled(t, M, t') = ({}^\bullet t \leq M - {}^\bullet t' + t'^\bullet) \wedge ((t = t') \vee ({}^\bullet t > M - {}^\bullet t'))$$

De la même façon, t est dite *désensibilisée* par le tir de t' à partir du marquage M, et nous notons $disabled(t, M, t')$, si t est sensibilisée par M mais ne l'est plus par $M - {}^\bullet t'$.

Par extension, nous notons $enabled(M, t')$ (respectivement $disabled(M, t')$) l'ensemble des transitions nouvellement sensibilisées (respectivement désensibilisées) par le tir de t' depuis M.

Maintenant, nous pouvons définir la sémantique des réseaux de Petri T-temporels sous la forme d'un système de transitions temporisé[2] (TTS).

Définition 6 (Sémantique d'un TPN) *La sémantique d'un réseau de Petri T-temporel* $\mathcal{T}$ *est définie sous la forme d'un système de transitions temporisé* $\mathcal{S}_\mathcal{T} = (Q, q_0, \rightarrow)$ *tel que :*

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$ *;*
- $q_0 = (M_0, \overline{0})$ *;*
- $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ *est la relation de transition incluant des transitions continues et des transitions discrètes :*
  - *la relation de transition* continue *est définie* $\forall d \in \mathbb{R}^+$ *par :*

$$(M, \nu) \xrightarrow{d} (M, \nu') \text{ ssi } \begin{cases} \nu' = \nu + d, \\ \forall t_k \in T, M \geq {}^\bullet t_k \Rightarrow \nu'(t_k) \leq \beta(t_k). \end{cases}$$

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

$$(M, \nu) \xrightarrow{t_i} (M', \nu') \text{ ssi } \begin{cases} M \geq {}^\bullet t_i, \\ \alpha(t_i) \leq \nu(t_i) \leq \beta(t_i), \\ M' = M - {}^\bullet t_i + t_i^\bullet, \\ \forall t_k, \nu'(t_k) = \begin{cases} 0 \text{ si } \uparrow enabled(t_k, M, t_i), \\ \nu(t_k) \text{ sinon.} \end{cases} \end{cases}$$

[2] voir 3.1.1.

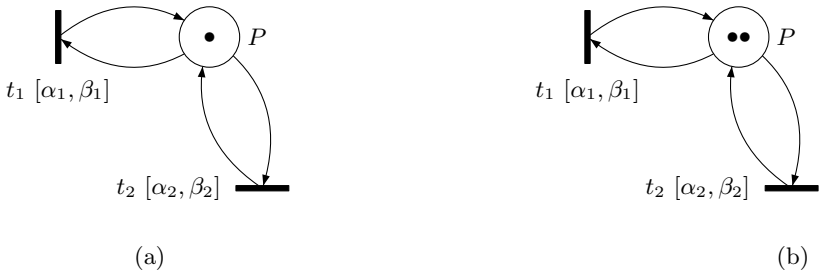

FIG. 4.2 – Example of newly enabled transitions

Quand une transition discrète est possible depuis l'état $s = (M, \nu)$ (de la sémantique) du réseau, nous dirons que la transition correspondante est *tirable*. Formellement :

Définition 7 (Transition tirable) *Soit $s = (M, \nu)$ un état de la sémantique d'un réseau de Petri T-temporel. Une transition t est dite* tirable *dans s si $M \geq {}^\bullet t$ et $\alpha(t) \leq \nu(t) \leq \beta(t)$.*

Nous pouvons remarquer que dans cette sémantique, si une place contient plusieurs jetons pouvant sensibiliser une ou des transitions sortantes, seul le nombre de jetons indiqué sur l'arc sera considéré. Les autres seront utilisés pour les tirs suivants éventuels des transitions sortantes. C'est une hypothèse *monoserveur*. Cette sémantique est illustrée par la figure 4.2 :

Supposons que t_1 est tirable. Sur la figure 4.2a, t_1 et t_2 sont sensibilisées par le marquage M et par le marquage $M' = M - {}^\bullet t_1 + t_1{}^\bullet$ mais pas par $M - {}^\bullet t_1$. Les transitions t_1 et t_2 sont donc *nouvellement* sensibilisées par le tir de t_1.

Sur la figure 4.2b, t_1 et t_2 sont sensibilisées par le marquage M et par le marquage $M' = M - {}^\bullet t_1 + t_1{}^\bullet$, mais aussi par $M - {}^\bullet t_1$. Dans ce cas, t_1 est *nouvellement* sensibilisée par le tir de t_1 (car elle est la transition tirée) mais pas t_2 : t_2 *reste* sensibilisée.

On peut assez facilement complexifier la sémantique de façon à ce qu'une transition puisse être sensibilisée plusieurs fois simultanément [Ber01]. On représente alors la transition ainsi *multisensibilisée* par autant de transitions « clonées » différentes que de sensibilisations. Ces clones ont tous les mêmes paramètres : arcs entrants et sortants et intervalle de tir. Si plusieurs transitions clonées sont tirables simultanément, on peut choisir celle à tirer de façon aléatoire ou en fonction de la durée de sensibilisation.

4.2.1 Problèmes intéressants

Étant donné un réseau de Petri T-temporel $\mathcal{T}$, plusieurs problèmes « intéressants[3] » peuvent être étudiés, notamment :

- l'*accessibilité de marquages* : étant donné un marquage M, « $\exists (M', \nu') \in \mathcal{S}_\mathcal{T}, M = M'$ » ;
- la *bornitude* :« $\exists b \in \mathbb{N}, \forall (M, \nu) \in \mathcal{S}_\mathcal{T}, \forall p \in P, M(p) \leq b$ » ;
- la *k-bornitude* : étant donné $k \in \mathbb{N}$, « $\forall (M, \nu) \in \mathcal{S}_\mathcal{T}, \forall p \in P, M(p) \leq k$ » ;
- l'*accessibilité d'états* : étant donné un état s, « $s \in \mathcal{S}_\mathcal{T}$ » ;
- la *vivacité* : « $\forall t \in T, \forall s \in \mathcal{S}_\mathcal{T}, \exists \sigma \in T^*, s' \in \mathcal{S}_\mathcal{T}, s \overset{\sigma.t}{\rightarrow} s'$ ».

L'accessibilité de marquages est un problème indécidable [JLL77], ce qui implique que la bornitude, l'accessibilité d'états et la vivacité sont également des problèmes indécidables. Par contre, la k-bornitude est décidable par exemple en utilisant la construction du graphe des classes d'états de [BD91] présentée dans la section suivante.

[3] À savoir non triviaux et qui apportent de l'information sur le modèle.

Dans le cas des réseaux de Petri T-temporels bornés, l'accessibilité de marquages, l'accessibilité d'états et la vivacité sont décidables.

Pour tenter de résoudre ces problèmes, nous pouvons essayer de construire l'espace d'états du réseau. Cependant, nous voyons très facilement que la sémantique que nous avons donnée implique que le modèle a un espace d'états infini. Non seulement le réseau n'est pas forcément *borné*, c'est à dire que son nombre de marquage n'est pas fini, mais surtout le temps est *dense* donc les horloges peuvent prendre un nombre infini de valeurs. Pour calculer l'espace d'états, il faut donc recourir à une méthode *symbolique*.

4.3 La méthode du graphe des classes d'états

La méthode la plus courante de calcul de l'espace d'états d'un réseau de Petri T-temporel est le calcul du graphe des classes d'états [Men82, BD91, Ber01]. Puisque l'espace d'états est infini, il faut regrouper les états en un nombre fini[4] de groupes. Dans cette méthode les groupes sont les *classes d'états*.

4.3.1 Classes d'états

Intuitivement, une classe d'états regroupe tous les états atteints par une même séquence de tir faisable du réseau. En particulier, tous les états d'une même classe ont donc le même marquage.

Définition 8 (Classe d'états) *Une* classe d'états *C, d'un réseau de Petri T-temporel, est un couple (M, D) où M est un marquage du réseau et D un polyèdre appelé* domaine de tir.
Les inéquations de D sont de deux types [BD91]

$$\begin{cases} \alpha_i \leq \theta_i \leq \beta_i \ (\forall i \text{ tel que } t_i \text{ est sensibilisée}), \\ -\gamma_{kj} \leq \theta_j - \theta_k \leq \gamma_{jk}, \ \forall j,k \text{ tels que } j \neq k \text{ and } (t_j, t_k) \in enabled(M) \times enabled(M) \end{cases}$$

θ_i représente l'instant de tir de la transition sensibilisée t_i relativement *à l'instant où l'on est entré dans la classe.*

Une conséquence de cette définition est que les domaines de tir des classes d'états peuvent être exprimés sous la forme de matrices de différences[5] (DBM). Cela permet donc une manipulation assez efficace[6].

4.3.2 Successeurs d'une classe d'états

Nous allons maintenant définir le successeur d'une classe d'états par une transition tirable. Nous définissons donc ce que l'on entend par transition *tirable depuis une classe d'états*.

Définition 9 (Transition tirable depuis une classe d'états) *Soit une classe d'états $C = (M, D)$. Une transition t_i est dite* tirable *depuis C si t_i est sensibilisée par M et il existe une solution $(\theta_1^*, \ldots, \theta_n^*)$ de D telle que $\forall j \in [\![1, n]\!]\{i\}, \theta_i^* \leq \theta_j^*$.*

[4] Sous certaines conditions, voir 4.3.4.
[5] voir 3.2.
[6] De complexité polynomiale.

Autrement dit, il existe un état (de la sémantique du réseau) s contenu dans la classe C tel que t est tirable dans s.

Soit une classe $C = (M, \nu)$ et une transition tirable t_f. La classe $C' = (M', D')$, successeur de C par t_f est donnée par :

- Le nouveau marquage est calculé de façon classique par $M' = M - {}^\bullet t_f + t_f{}^\bullet$
- Le nouveau domaine de tir, D', est calculé selon les étapes suivantes. Nous le noterons $next(D, t_f)$
  1. changements de variables $\forall j, \theta_j = \theta_f + \theta'_j$;
  2. $\forall j \neq t_f$, ajout des contraintes $\theta'_j \geq 0$;
  3. élimination des variables correspondant à des transitions désensibilisées par le tir de t_f (ce qui inclut donc t_f), par exemple en utilisant la méthode de Fourier-Motzkin [Dan63] ;
  4. ajout des inéquations relatives aux transitions nouvellement sensibilisées par le tir de t_f :
  $$\forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq \theta'_k \leq \beta(t_k).$$
  5. détermination la forme canonique D'^* du nouveau domaine de tir.

Les changements de variables modélisent l'écoulement du temps pour les transitions sensibilisées, par un changement d'origine du temps : la nouvelle origine devient l'instant de tir de t_f.

Les contraintes $\theta'_j \geq 0$ peuvent également être écrites $\theta_j \geq \theta_f$, ce qui exprime le fait que nous avons choisi de tirer t_f et, par conséquent, toutes les autres transitions sensibilisées seront tirées plus tard.

4.3.3 Graphe des classes d'états

Pour calculer le graphe des classes d'un réseau de Petri T-temporel $\mathcal{T}$, il suffit de calculer itérativement tous les successeurs de la classe initiale par la relation que nous venons de définir, la classe initiale C_0 étant définie par $C_0 = (M_0, D_0)$, où M_0 est le marquage initial du réseau et $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$. Nous obtenons alors le système de transitions potentiellement infini $\Gamma'(\mathcal{T}) = (\mathbf{C}, C_0, \rightarrow)$ où

- $\mathbf{C} = \mathbb{N}^P \times \mathbb{R}^T$;
- $C_0 = (M_0, D_0)$, où M_0 est le marquage initial du réseau et $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$;
- $\rightarrow \in \mathbf{C} \times T \times \mathbf{C}$ est la relation de transition définie par :
$$(M, D) \xrightarrow{t} (M', D') \text{ ssi } \begin{cases} M' = M - {}^\bullet t + t^\bullet, \\ t \text{ est tirable depuis } (M, D), \\ D' = next(D, t). \end{cases}$$

Comme nous l'avons dit, ce système de transitions est potentiellement infini car il n'y a *a priori* aucune raison pour que l'on finisse toujours par atteindre des classes sans successeur. Nous allons donc définir un critère de convergence sous la forme d'une relation d'équivalence entre les classes :

Définition 10 (Egalité de classes d'états) *Deux classes d'états $C = (M, D)$ et $C' = (M', D')$ sont dites* égales *si $M = M'$ et $[\![D]\!] = [\![D']\!]$. Nous notons $C \equiv C'$.*

Nous pouvons alors définir le graphe des classes $\Gamma(\mathcal{T})$ comme le *quotient* de $\Gamma'(\mathcal{T})$ par la relation d'égalité : $\Gamma(\mathcal{T}) = \Gamma'(\mathcal{T}) / \equiv$.

4.3.4 Finitude

Étant donné un réseau de Petri T-temporel, le problème est de savoir si le nombre de ses classes d'états est fini. Pour résoudre ce problème, Berthomieu et Diaz ont démontré la condition nécessaire et suffisante suivante [BD91] :

Théorème 1 *Le nombre de classes du graphe des classes d'un réseau de Petri T-temporel $\mathcal{T}$ est fini si et seulement si $\mathcal{T}$ est borné.*

Par conséquent, ce problème est indécidable car le problème de bornitude d'un réseau de Petri T-temporel est indécidable. Bernard Berthomieu et Michel Diaz proposent donc également la condition suffisante de bornitude suivante [BD91] :

Théorème 2 *Un réseau de Petri T-temporel est borné si son réseau non temporisé sous-jacent est borné et si les instants de tir au plus tôt et au plus tard des transitions sont des nombres rationnels.*

Cette condition suffisante est en général trop restrictive. C'est-à-dire qu'elle n'est pas vérifiée pour de nombreux réseaux qui sont pourtant bornés. C'est pourquoi il est préférable d'utiliser les conditions nécessaires de non bornitude suivantes pendant l'exécution de l'algorithme de calcul du graphe des classes [BD91] :

Théorème 3 *Un réseau de Petri T-temporel est borné s'il n'existe pas de paire de classes d'états $C = (M, D)$ et $C' = (M', D')$, accessibles depuis la classe initiale telle que*

1. *C' est accessible depuis C ;*
2. *$M' > M$;*
3. *$[\![D']\!] = [\![D]\!]$;*
4. *$\forall p \in \{p \in P, M'(p) > M(p)\}, M(p) > \max_{t \in T} {}^\bullet t(p)$.*

Les conditions 1 et 2 constituent déjà une condition nécessaire de non bornitude. Avec la condition 3, cette condition est plus forte et seul un petit nombre de réseaux bornés devraient vérifier ces quatre conditions. Cependant, il ne faut pas négliger le coût du calcul des conditions 3 et 4.

4.3.5 Exemple

La figure 4.3 présente le graphe des classes du réseau de la figure 4.1. Nous avons annoté le graphe avec les marquages des différentes classes ainsi que les domaines des deux premières classes.

4.3.6 Vérification avec le graphe des classes d'états

Langage

Le graphe des classes donne l'ensemble des marquages du réseau et préserve son langage *non temporisé*. C'est-à-dire qu'il fournit l'ordre d'apparition des événements (tir de transition ou occurrence d'un marquage). Il permet donc de vérifier des propriétés de type *Linear Temporal Logic* (LTL) [Pnu77]. Cependant, nous pouvons remarquer que si la classe C' est le successeur par la transition t de la classe C, cela n'implique pas forcément que tous les états de C ont un successeur par t dans C' mais uniquement qu'il *existe* au moins un état de C qui

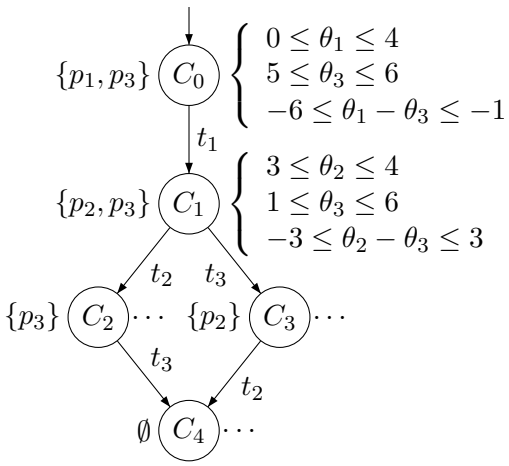

FIG. 4.3 – Exemple de graphe des classes

possède un successeur par t dans C'. C'est pourquoi nous ne pouvons pas vérifier des propriétés de type *Computation Tree Logic* (CTL) [Pnu77] sur le graphe des classes. Pour remédier à cela, Berthomieu et Vernadat ont proposé une construction de classes dites *atomiques* [BV03] telle que le graphe des classes atomiques préserve les propriétés de branchement du réseau de Petri permettant ainsi la vérification de propriétés CTL.

Cependant, le graphe des classes (même atomiques) ne fournit pas d'information quantitative sur les instants auxquels les tirs des transitions se sont produits : regardons à nouveau les figures 4.1 et 4.3. Si nous tirons t_1 à l'instant 0, alors la suite ne peut être que le tir de t_2 suivi du tir de t_3 car les deux intervalles de tir de ces transitions sont disjoints. Par contre, si nous tirons t_1 après 4 unités de temps, la séquence forcée est t_3 suivie de t_2. Cette information n'apparaît pas dans la classe C_1 du graphe des classes d'états, même si nous ajoutons l'information contenue dans le domaine de tir.

Afin de vérifier des propriétés temporelles avec le graphe des classes, la méthode la plus courante consiste à avoir recours à des *observateurs*.

Observateurs

Un observateur [TSLT97] est un réseau de Petri T-temporel ajouté au réseau qu'il observe de façon *non intrusive*. La valeur de vérité de la propriété modélisée par l'observateur est alors donnée par l'occurrence ou non d'un marquage ou du tir d'une transition dans l'espace d'états du système « réseau de Petri T-temporel observé + observateur ». Cela est facilement vérifiable par le calcul du graphe des classes d'états de ce système.

Par exemple, l'observateur de la figure 4.4, représenté en traits interrompus, modélise pour ce réseau sauf, la propriété « le temps de séjour dans la place p_2 est toujours inférieur ou égal à 5 unités de temps ». Cette propriété est trivialement vraie, ce qui se traduit par l'absence de tir de t_{obs} dans le graphe des classes d'états du réseau avec l'observateur.

L'utilisation des observateurs n'est pas sans conséquence. D'une part, la taille du système augmente avec chaque observateur ajouté. Quand les propriétés à vérifier portent sur des occurrences de marquages, l'observateur peut être aussi gros que le réseau observé lui-même. Étant donné le problème de l'explosion combinatoire, cette augmentation de taille ne doit pas

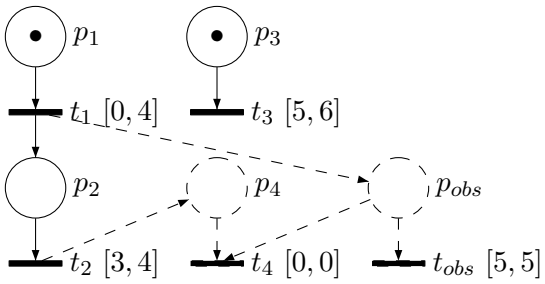

FIG. 4.4 – Un réseau de Petri T-temporel et un observateur

être négligée. D'autre part, chaque propriété requiert un observateur spécifique et donc un nouveau calcul de graphe des classes d'états. Si toutes les propriétés à vérifier sont connues à l'avance, tous les observateurs peuvent être ajoutés d'un seul coup, ce qui permet de ne faire qu'un seul calcul de graphe des classes. Cependant, cela n'est pas forcément avantageux si la taille du système s'en trouve trop augmentée.

Pour l'utilisation des observateurs, seuls nous intéressent en général, l'ensemble des marquages et - ou - l'ensemble des transitions tirées. Dans ce cas, nous pouvons nous contenter d'arrêter le calcul des successeurs lorsque la classe (M', D') que nous venons de calculer est incluse, au sens de la définition 11, dans une classe précédemment calculée (M, D), plutôt qu'égale.

Définition 11 (Inclusion de classes d'états) *Une classe d'états $C = (M, D)$ est dite* incluse *dans une classe d'états $C' = (M', D')$ si $M = M'$ et $[\![D]\!] \subset [\![D']\!]$. Nous notons $C \subset C'$.*

Si nous avons $M' = M$ mais pas $[\![D']\!] \subset [\![D]\!]$, nous pouvons tout de même boucler sur la classe (M, D) qui devient alors $(M, D \cup D')$ et continuer l'exploration pour les états de $(M, D' - (D' \cap D))$. Nous obtenons ainsi le graphe des marquages du réseau. Bien sûr le langage non temporisé du réseau est perdu.

4.4 Conclusion

Dans ce chapitre, nous avons présenté les réseaux de Petri T-temporels ainsi que la méthode la plus courante de calcul de l'espace d'états de ce modèle : le graphe des classes d'états. Nous avons vu que le graphe des classes d'états donne l'ensemble des marquages du réseau ainsi que les séquences non temporisées de transitions. Afin de vérifier des propriétés temporelles, il faut avoir recours aux observateurs. Cela présente deux inconvénients principaux : la taille du système est augmentée pour chaque observateur ajouté, ce qui le rend plus sujet au phénomène d'explosion combinatoire. Et chaque propriété à vérifier requiert un observateur spécifique et donc un calcul de graphe des classes supplémentaire.

Dans le chapitre suivant, nous présentons une méthode pour calculer l'espace d'états sous la forme d'un automate temporisé ce qui permet notamment d'obtenir directement le langage temporisé du réseau de Petri T-temporel et donc de s'affranchir des problèmes liés à ces observateurs complexes.

Chapitre 5

Automate temporisé des classes d'états

Dans ce chapitre, nous proposons une méthode, publiée dans [LR03b] et étendue dans [LR], pour calculer l'espace d'états d'un réseau de Petri T-temporel borné sous la forme d'un automate temporisé. Nous prouvons la correction du calcul par une bisimulation temporelle entre le réseau de Petri et sa traduction sous forme d'automate temporisé. Le nombre d'horloges de l'automate temporisé obtenu est minimal en un sens que nous préciserons, ce qui nous permet de vérifier efficacement des propriétés temps réel sur les réseaux de Petri T-temporels avec des outils reconnus comme Uppaal [LPY97] et Kronos [Yov97]. Nous proposons enfin une méthode pour exprimer des propriétés en utilisant la logique temporelle *Timed Computation Tree Logic* (TCTL) [ACD93] directement sur le réseau de Petri T-temporel et de les vérifier grâce à l'automate temporisé obtenu.

Réfutons tout de suite une objection légitime, à savoir : puisque finalement nous obtenons un automate temporisé, pourquoi ne pas modéliser directement le système avec ce formalisme ?

D'une part, il est clair que certains systèmes sont plus facilement exprimables sous la forme d'un réseau de Petri T-temporel que sous la forme d'un automate temporisé : l'exemple le plus simple est peut-être celui d'un sémaphore non borné, ou de borne non connue. D'autre part, cette méthode constitue une passerelle entre les deux modèles et permet ainsi de modéliser une partie du système sous la forme d'un automate temporisé et l'autre sous la forme d'un réseau de Petri T-temporel, permettant ainsi d'utiliser les deux modèles là où ils sont le plus efficaces.

Dans un premier temps nous allons présenter les automates temporisés. Puis nous expliciterons la méthode de calcul de l'automate temporisé des classes d'états d'un réseau de Petri T-temporel. Nous donnerons ensuite des propriétés intéressantes de cet automate et enfin nous montrerons comment vérifier des propriétés exprimées en TCTL directement sur un réseau de Petri T-temporel.

5.1 Automates temporisés

5.1.1 Présentation

Les automates temporisés, ou *timed automata* (TA) en anglais, ont été introduits par Alur et Dill en 1994 [AD94] et étendus avec la notion d'invariant par Henzinger *et al.* dans

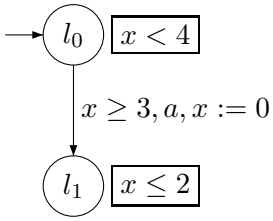

FIG. 5.1 – Un automate temporisé

[HNSY94]. Le temps est ajouté au modèle classique des automates sous la forme d'horloges et de prédicats sur ces horloges. Ces prédicats sont de deux types : les *gardes*, associées aux transitions discrètes, donnent des contraintes sur les horloges à respecter pour pouvoir exécuter cette transition discrète. Les *invariants* associés aux localités, donnent des contraintes qui doivent être respectées pour que l'on puisse laisser passer du temps dans les localités.

La figure 5.1 donne un exemple très simple d'automate temporisé. La localité initiale est l_0. L'automate dispose d'une seule horloge : x. x est nul à l'instant initial et l'invariant de l_0 indique donc que l'on pourra rester dans l_0 strictement moins de 4 unités de temps. Dès que 3 unités de temps se sont écoulées, la garde de la transition entre l_0 et l_1 est vérifiée (et l'invariant de l_1 est vérifié après franchissement de cette transition) et la transition peut donc être franchie. Si cette transition est franchie, l'horloge x est remise à zéro et la localité active devient l_1.

5.1.2 Définitions

Définition 12 (Automate temporisé) *[HNSY94] Un* automate temporisé *est un 6-uplet* (L, l_0, X, A, E, Inv) *où*

- L *est un ensemble fini de* localités *;*
- l_0 *est la* localité initiale *;*
- X *est un ensemble fini d'*horloges *à valeurs réelles positives ;*
- A *est un ensemble fini d'*actions *;*
- $E \subset L \times \mathcal{C}(X) \times A \times 2^X \times 2^{X^2} \times L$ *est un ensemble fini d'*arêtes*. Soit* $e = (l, \delta, \alpha, R, \rho, l') \in E$*. e est l'arête reliant la localité* l *à la localité* l'*, avec la* garde δ*, l'action* α*, l'ensemble d'horloges à remettre à zéro* R *et la fonction d'affectation d'horloges* ρ*.*
- $Inv \in \mathcal{C}(X)^L$ *associe un* invariant *à chaque localité.*

La fonction ρ ne fait pas partie de la définition classique des automates temporisés mais son introduction ne change pas la complexité des problèmes associés à ce modèle.

Nous définissons la sémantique des automates temporisés sous la forme d'un système de transitions temporisé.

Définition 13 (Sémantique d'un automate temporisé) *La sémantique d'un automate temporisé* $\mathcal{A}$ *est définie sous la forme d'un système de transitions temporisé* $\mathcal{S}_\mathcal{A} = (Q, Q_0, \rightarrow)$ *où*

- $Q = L \times (\mathbb{R}^+)^X$ *;*
- $Q_0 = (l_0, \overline{0})$ *;*
- $\rightarrow \in Q \times (A \cup \mathbb{R}) \times Q$ *est la relation définie pour* $a \in A$ *et* $d \in \mathbb{R}^+$*, par :*

– *la relation de transition* discrète *:* $(l,\nu) \xrightarrow{a} (l',\nu')$ *ssi* $\exists (l,\delta,a,R,\rho,l') \in E$ *tel que*
$\begin{cases} \delta(\nu) = \texttt{true}, \\ \nu' = \nu[R \leftarrow 0][\rho], \\ Inv(l')(\nu') = \texttt{true} \end{cases}$

– *la relation de transition* continue *:* $(l,\nu) \xrightarrow{\epsilon(d)} (l,\nu')$ *ssi* $\begin{cases} \nu' = \nu + d, \\ \forall d' \in [0,d], Inv(l)(\nu + d') = \texttt{true} \end{cases}$

5.2 Automate temporisé des classes d'états

Nous proposons maintenant une extension du graphe des classes d'états présenté au chapitre précédent qui permet d'obtenir un automate temporisé à la place d'un simple graphe. Cela nous permet de vérifier des propriétés temporelles sur les réseau de Petri T-temporels. Par ailleurs, la méthode que nous proposons met l'accent sur la minimisation du nombre d'horloges de l'automate temporisé obtenu.

5.2.1 Classes d'états étendues

La construction des classes d'états du chapitre précédent est basée sur un temps relatif : dans le domaine de tir d'une classe d'états, on contraint les instants de tir des transitions sensibilisées exprimés par rapport à la date d'entrée dans la classe. Les valuations des états de la sémantique du réseau de Petri T-temporel (définition 6) sont « oubliées ». Nous allons les rendre à nouveau explicites à l'aide des horloges de l'automate temporisé que nous allons construire.

Intuitivement, nous pouvons donc voir immédiatement qu'il est suffisant d'associer une horloge à chaque transition. Cependant, dans un souci d'optimisation du nombre d'horloges, nous allons déterminer pour chaque classe d'états calculée le nombre d'horloges nécessaires pour exprimer les valuations des transitions sensibilisées. Nous définissons donc des *classes d'états étendues* de la façon suivante :

Définition 14 (Classe d'états étendues) *Une classe d'états étendue, d'un réseau de Petri T-temporel, est un 4-uplet* $(M, D, \chi, trans)$*, où* M *est un marquage,* D *un polyèdre appelé* domaine de tir*,* χ *un ensemble d'*horloges *et* $trans \in (2^T)^\chi$ *associe à chaque horloge un ensemble de transitions.*

Nous avons donc ajouté deux composantes. χ représente les horloges nécessaires pour exprimer les valuations des transitions sensibilisées par M. La fonction $trans$ désigne pour chaque horloge les transitions dont elle représente la valuation. Par conséquent, pour toute transition t, l'image réciproque de t par $trans$ est réduite à un singleton. Nous le notons $trans^{-1}(t)$.

5.2.2 Successeur d'une classe d'états étendue

Le calcul des successeurs diffère de celui du chapitre précédent par les calculs requis pour les deux composantes supplémentaires. Le calcul du nouveau marquage et celui du nouveau domaine de tir restent inchangés. Soit une classe d'états étendue $C = (M, D, \chi, trans)$ et une transition tirable t_f. Pour calculer la classe d'états étendue $C' = (M', D', \chi', trans')$ obtenue par le tir de t_f depuis C, nous avons les étapes supplémentaires suivantes :

1. pour chaque horloge x de χ, les transitions désensibilisées par le tir de t_f sont retirées de $trans(x)$;
2. les horloges dont l'image par $trans$ est vide sont retirées de χ ;
3. s'il y a des transitions nouvellement sensibilisées par le tir de t_f, deux cas sont possibles :
   - il existe une horloge x dont la valeur est 0. Alors, les transitions nouvellement sensibilisées sont ajoutées à $trans(x)$,
   - il n'existe pas d'horloge nulle. Alors nous créons une nouvelle horloge x_i associée aux transitions nouvellement sensibilisées. L'indice i choisi est le plus petit indice disponible parmi ceux des horloges de χ. Nous ajoutons x_i à χ et $trans(x_i)$ est l'ensemble des transitions nouvellement sensibilisées.

Le calcul de l'automate temporisé des classes d'états se fait en deux étapes. Dans la première nous allons calculer un graphe des classes d'états étendues de façon assez similaire au graphe des classes d'états classiques. Puis nous déduirons syntaxiquement l'automate temporisé de ce graphe.

5.2.3 Graphe des classes d'états étendues

La classe d'états étendue initiale est $C_0 = (M_0, D_0, \chi_0, trans_0)$, où M_0 est le marquage initial, $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$, $\chi_0 = \{x_0\}$ et $trans_0 = (x_0, enabled(M_0))$.

Alors, pour le réseau de Petri T-temporel $\mathcal{T}$, nous itérons le calcul des successeurs depuis la classe d'états étendue initiale, comme pour les classes d'états classique.

Nous obtenons ainsi le système de transitions potentiellement infini $\Delta''(\mathcal{T}) = (\mathbf{C}^{ext}, C_0, \rightarrow^{ext})$ où :

- $\mathbf{C}^{ext} = \mathbb{N}^P \times \mathbb{R}^T \times 2^X \times (2^T)^X$, X étant l'ensemble de toutes les horloges utilisées,
- $C_0 = (M_0, D_0, \chi_0, trans_0)$, où M_0 est le marquage initial, $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$, $\chi_0 = \{x_0\}$ et $trans_0 = (x_0, enabled(M_0))$
- $\rightarrow^{ext} \in C^{ext} \times T \times C^{ext}$ est la relation de transition définie par :

$(M, D, \chi, trans) \overset{t}{\rightarrow}^{ext} (M', D', \chi', trans')$ ssi

$$\begin{cases} t \text{ est tirable depuis } (M, D), \\ M' = M - {}^\bullet t + t^\bullet, \\ D' = next(D, t), \\ \text{soit } DC = \{x \in \chi, trans(x) - disabled(M, t) = \emptyset\}, \\ \text{si } \uparrow enabled(M, t) = \emptyset, \text{ alors } \chi' = \chi - DC \text{ et } \forall x \in \chi', trans'(x) = trans(x), \\ \text{sinon} \begin{cases} \text{si } \exists x_j \in \chi \text{ t.q. } x_j = 0, \text{ alors } \chi' = \chi - DC, \\ \begin{cases} \forall x \in \chi' - \{x_j\}, trans'(x) = trans(x), \\ trans'(x_j) = trans(x_j) \cup \uparrow enabled(M, t), \end{cases} \\ \text{sinon} \begin{cases} i = \min\{k \in \mathbb{N} | x_k \notin \chi - DC\}, \chi' = \chi - DC \cup \{x_i\}, \\ \forall x \in \chi' - \{x_i\}, trans'(x) = trans(x), \\ trans'(x_i) = \uparrow enabled(M, t), \end{cases} \end{cases} \end{cases}$$

Nous pouvons définir le critère de convergence pour le graphe des classes d'états étendues comme l'égalité de la définition 15.

Définition 15 (Egalité de classes d'états étendues) *Deux classes d'états étendues $C = (M, D, \chi, trans)$ et $C' = (M', D', \chi', trans')$ sont dites* égales *si $M = M'$, $[\![D]\!] = [\![D']\!]$, $\chi = \chi'$ et $trans = trans'$. Nous notons $C \equiv C'$.*

Cependant, la structure finale d'automate temporisé que nous allons obtenir permet d'utiliser une relation d'équivalence plus large tout en conservant le langage (temporisé) du réseau de Petri initial. Nous l'appelons *clock-similarité*.

Définition 16 (Clock-similarité) *Deux classes d'états étendues* $C = (M, D, \chi, trans)$ *et* $C' = (M', D', \chi', trans')$ *sont* clock-similaires*, ce que nous notons* $C \approx C'$*, si elles ont le même marquage, le même nombre d'horloges et si leurs horloges sont associées aux mêmes transitions :*

$$C \approx C' \Leftrightarrow \begin{cases} M = M', \\ |\chi| = |\chi'|, \\ \forall x \in \chi, \exists x' \in \chi', trans(x) = trans'(x'). \end{cases}$$

Le nom des horloges n'est pas important dans la mesure où l'on dispose des affectations d'horloges pour nos automates temporisés. On peut ainsi renommer les horloges de façon adéquate lors d'une convergence. Le point clé est que $trans$ et $trans'$ doivent partitionner l'espace des transitions de la même manière. L'égalité des marquages et du nombre d'horloges est une conséquence de cette propriété. Nous reviendrons sur l'intérêt de cette propriété après avoir donné la définition de l'automate temporisé des classes d'états.

D'un point de vue théorique, nous pouvons définir le graphe des classes d'états étendues $\Delta'(\mathcal{T})$ d'un réseau de Petri T-temporel $\mathcal{T}$ comme le quotient de $\Delta''(\mathcal{T})$ par la relation de *clock*-similarité : $\Delta'(\mathcal{T}) = \Delta''(\mathcal{T})/\approx$.

D'un point de vue pratique, et au contraire de la relation d'égalité, nous ne pouvons pas systématiquement arrêter le calcul des successeurs quand nous rencontrons une classe qui est *clock*-similaire à une classe déjà calculée. En effet, nous n'avons aucune contrainte sur le domaine de tir. Il se peut donc que les deux classes n'aient pas les mêmes successeurs. Par conséquent, nous définissons un critère d'arrêt sous la forme d'une inclusion de domaine de tir.

Définition 17 (Inclusion de classes d'états étendues) *Une classe d'états étendue* $C' = (M', D', \chi', trans')$ *est* incluse *dans une classe d'états étendue* $C = (M, D, \chi, trans)$ *si* C *et* C' *sont clock-similaires et* $[\![D']\!] \subset [\![D]\!]$*. Nous notons* $C' \subset C$*.*

Quand la classe calculée $C = (M, D, \chi, trans)$ est incluse dans une classe déjà générée $C' = (M', D', \chi', trans')$, nous pouvons arrêter le calcul des successeurs pour cette branche du graphe. Si nous avons une « simple » *clock*-similarité, nous convergeons mais nous continuons le calcul des successeurs pour la classe d'états étendue $(M, D - D \cap D', \chi, trans)$. Par ailleurs, le domaine de tir de D' devient $D \cup D'$.

Ici encore, l'inclusion est suffisante pour conserver le langage du réseau de Petri T-temporel à cause de la structure d'automate temporisé du résultat de notre méthode. En effet, en convergeant par inclusion ce sont les valeurs des horloges ainsi que les gardes et les invariants qui vont interdire certaines transitions.

La figure 5.2 montre l'exemple du graphe des classes d'états étendues du réseau de la figure 4.1 du chapitre 4. Nous avons uniquement représenté les horloges et les transitions qui leur sont associées sous la forme $x : trans(x)$. Cet exemple est très simple, car le réseau ne présente pas de cycle et, ici, le critère de convergence *clock*-similarité donne le même résultat que l'égalité.

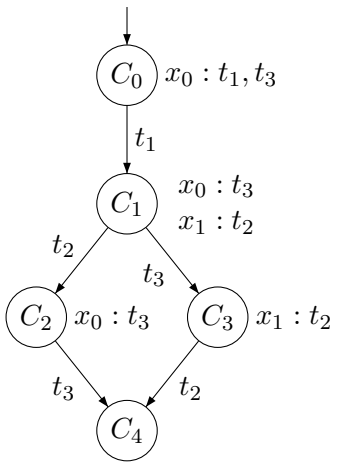

FIG. 5.2 – Graphe des classes étendues du TPN de la figure 4.1

5.2.4 Automate temporisé des classes d'états étendues

À partir du graphe des classes étendues $\Delta'(\mathcal{T})$ du réseau de Petri T-temporel borné $\mathcal{T}$, nous pouvons maintenant définir syntaxiquement l'*automate temporisé des classes d'états étendues* $\Delta(\mathcal{T})$:

Définition 18 (Automate temporisé des classes d'états étendues) *Soit un réseau de Petri T-temporel borné $\mathcal{T}$. L'*automate temporisé des classes d'états étendues $\Delta(\mathcal{T}) = (L, l_0, X, A, E, Inv)$ *est défini à partir du graphe des classes d'états étendues par :*

- *L, l'ensemble des localités, est l'ensemble des classes d'états étendues C^{ext} ;*
- *l_0 est la classe d'états étendue initiale $(M_0, D_0, \chi_0, trans_0)$;*
- *$X = \bigcup_{(M,D,\chi,trans)\in C^{ext}} \chi$*
- *$A = T$ est l'ensemble des transitions du réseau de Petri T-temporel*
- *E est l'ensemble des arêtes défini comme suit :*

$$\forall C_i = (M_i, D_i, \chi_i, trans_i), C_j = (M_j, D_j, \chi_j, trans_j) \in C^{ext},$$
$$\exists C_i \overset{t}{\rightarrow}^{ext} C_j \Leftrightarrow \exists (l_i, \delta, a, R, \rho, l_j) \ t.q. \begin{cases} \delta = ({trans_i}^{-1}(t) \geq \alpha(t)), \\ a = t, \\ R = {trans_j}^{-1}(\uparrow enabled(M_i, t)), \\ \forall x \in \chi_i, x' \in \chi_j, \\ \ t.q.\ trans_i(x) = trans_j(x') \\ \ et\ x' \notin R, \rho(x) = x' \end{cases}$$

- $\forall C_i \in C^{ext}, Inv(l_i) = \bigwedge_{x\in\chi_i, t\in trans_i(x)}(x \leq \beta(t))$.

L'automate temporisé des classes d'états étendues est obtenu à partir du graphe des classes *étendues* mais dans la suite de ce document nous l'appellerons simplement automate temporisé des classes d'états.

Si deux classes d'états étendues $C = (M, D, \chi, trans)$ et $C' = (M', D', \chi', trans')$ sont *clock*-similaires, alors comme nous l'avons vu, $trans$ et $trans'$ définissent les mêmes partitions de l'espace des transitions sensibilisées par M. Cette définition montre alors en particulier

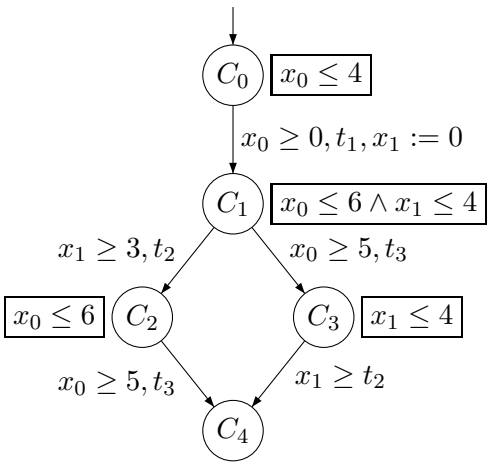

FIG. 5.3 – Automate temporisé des classes d'états du TPN de la figure 4.1

que les invariants et gardes des localités correspondant à C et C' sont les mêmes, au nom des horloges près. Par conséquent, la *clock*-similarité est une relation d'équivalence pertinente de ce point de vue, avec un renommage des horloges.

La figure 5.3 montre l'exemple de l'automate temporisé des classes d'états du réseau de la figure 4.1 du chapitre 4. On peut constater que l'automate temporisé ne possède que deux horloges alors que le réseau de Petri initial a, quant à lui, trois transitions.

L'exemple que nous venons de voir ne permet pas d'illustrer le critère de convergence particulier de l'algorithme. La figure 5.4 présente un autre exemple de réseau de Petri T-temporel ainsi que son graphe des classes d'états. Son graphe des classes étendues et son automate temporisé des classes d'états sont donnés sur la figure 5.5. À titre de comparaison, nous donnons également l'automate temporisé tel qu'il serait avec une horloge par transition du réseau de Petri T-temporel dans la figure 5.6

5.3 Propriétés

5.3.1 Bisimulation

Nous définissons une bisimulation [1] entre le réseau de Petri T-temporel borné $\mathcal{T}$ et son automate temporisé des classes d'états $\Delta(\mathcal{T})$. Cela prouve que le langage accepté par $\mathcal{T}$ est le même que le langage accepté par $\Delta(\mathcal{T})$. Cela nous permet de vérifier des propriétés exprimées avec TCTL sur le réseau de Petri T-temporel $\mathcal{T}$ [2].

Théorème 4 (Bisimulation) *Soient $Q_\mathcal{T}$ l'ensemble des états du réseau de Petri T-temporel $\mathcal{T}$ et $Q_\mathcal{A}$ l'ensemble des états de son automate temporisé des classes d'états $\mathcal{A} = (L, l_0, X, A, E, Inv)$ [3]. Soit $\mathcal{R} \subset Q_\mathcal{T} \times Q_\mathcal{A}$ une relation binaire telle que $\forall s = (M_\mathcal{T}, \nu_\mathcal{T}) \in Q_\mathcal{T}, \forall a = (l, \nu_\mathcal{A}) \in Q_\mathcal{A}, s\mathcal{R}a \Leftrightarrow M_\mathcal{T} = M_\mathcal{A}$ avec $M_\mathcal{A}$ le marquage de la classe d'états étendue l et $\forall t \in enabled(M_\mathcal{T}), \exists x \in X$ t.q. $trans^{-1}(t) = x$ et $\nu_\mathcal{T}(t) = \nu_\mathcal{A}(x)$.*

[1] Voir la sous section 3.1.4.

[2] Voir la section 5.4.

[3] Rappelons que l'ensemble L des localités de l'automate est l'ensemble C^{ext} des classes d'états étendues de $\mathcal{T}$.

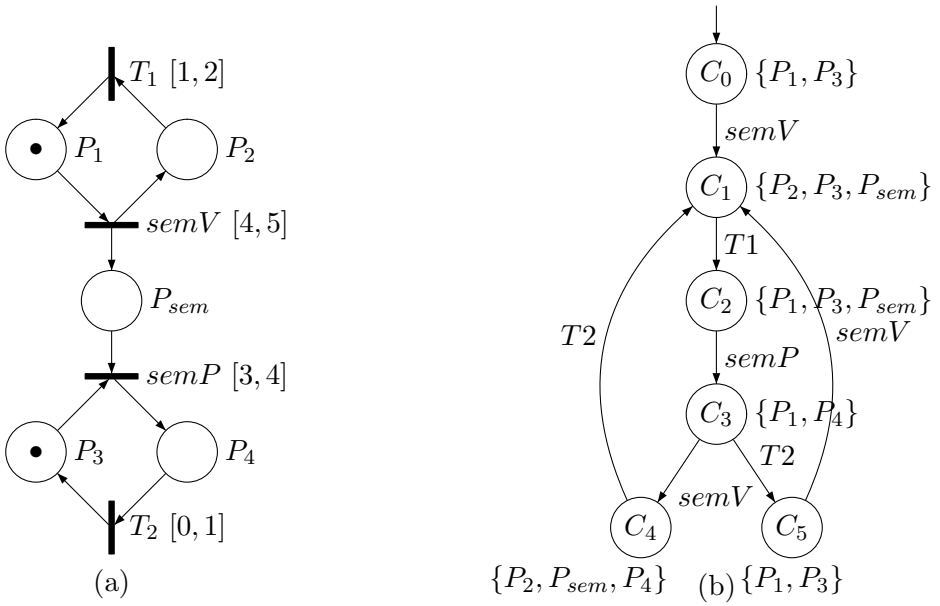

FIG. 5.4 – Deux tâches synchronisées par un sémaphore : modèle TPN (a) et graphe des classes (b)

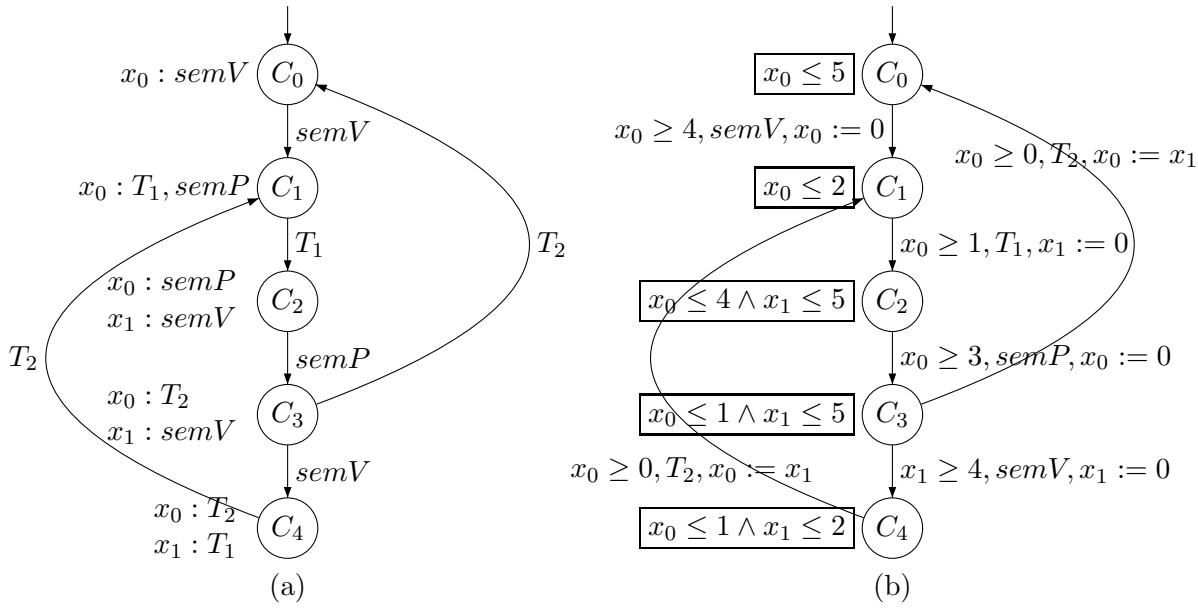

FIG. 5.5 – Deux tâches synchronisées par un sémaphore : graphe des classes étendues (a) et automate temporisé des classes (b)

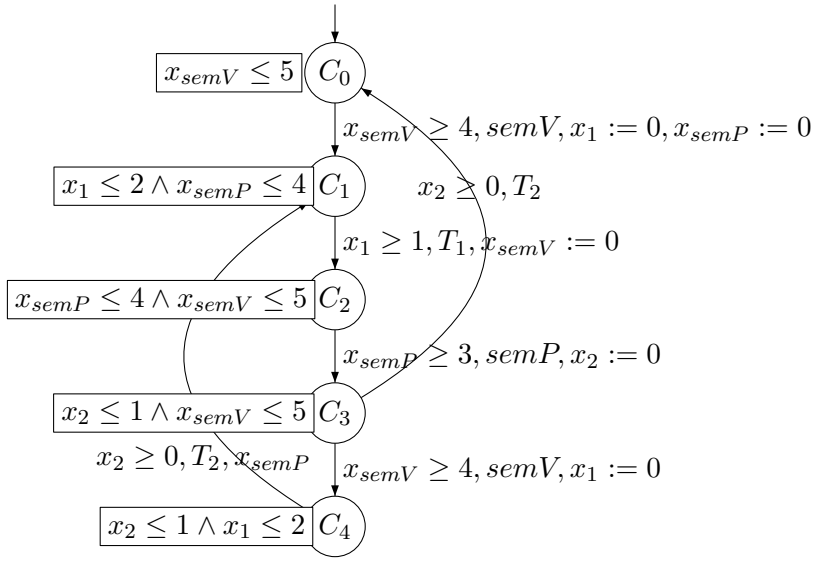

FIG. 5.6 – Automate temporisé avec une horloge par transition du TPN

$\mathcal{R}$ est une bisimulation.

Preuve. Soient $s = (M, \nu_{\mathcal{T}}) \in Q_{\mathcal{T}}$ un état du réseau de Petri T-temporel $\mathcal{T}$ et $a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}$ un état de son automate temporisé des classes d'états $\Delta(\mathcal{T})$ tels que $s\mathcal{R}a$.

– Considérons la transition continue $s \xrightarrow{\epsilon(d)} s'$, $d \in \mathbb{R}^+$, $s' = (M', \nu')$.

Prouvons d'abord que, pour tout $d \in \mathbb{R}^+$ tel que $s \xrightarrow{\epsilon(d)} s'$ est possible pour $\mathcal{T}$, $a \xrightarrow{\epsilon(d)} a'$ est également possible pour $\Delta(\mathcal{T})$.

En effet, $s \xrightarrow{\epsilon(d)} s'$ est équivalent à $\forall t \in enabled(M), \nu_{\mathcal{T}}(t) + d \leq \beta(t)$. Puisque $s\mathcal{R}a$, $\forall t \in enabled(M_{\mathcal{T}}), \exists x \in X$ t.q. $trans^{-1}(t) = x$ et $\nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x)$. Cela implique que $\forall t \in enabled(M), \exists x \in X$ t.q. $x = trans^{-1}(t)$ et $\nu_{\mathcal{A}}(x) + d \leq \beta(t)$, ce qui implique que nous pouvons laisser s'écouler le temps d en satisfaisant $Inv(l)$ sur l'automate $\Delta(\mathcal{T})$, car $Inv(l) = (\bigwedge_{x \in X} (x \leq \min_{t' \in trans(x)} \beta(t')))$. Donc $a \xrightarrow{\epsilon(d)} a'$ est possible.
Puisqu'il n'y a pas de changement de marquage entre s et s', les transitions sensibilisées restent les mêmes et par conséquent, les horloges sont les mêmes entre a et a'. De plus, si $a \xrightarrow{\epsilon(d)} a'$, alors $\nu'_{\mathcal{A}} = \nu_{\mathcal{A}} + d$ et si $s \xrightarrow{\epsilon(d)} s'$, alors $\nu'_{\mathcal{T}} = \nu_{\mathcal{T}} + d$. Donc, comme $s\mathcal{R}a$, $\forall t \in enabled(M'), \exists x \in X, \nu_{\mathcal{A}}(x) = \nu_{\mathcal{T}}(t)$. Nous avons donc $\forall t \in enabled(M'), \exists x \in X, \nu'_{\mathcal{A}}(x) = \nu_{\mathcal{T}}(x) + d = \nu'_{\mathcal{T}}(t)$. D'où, $s'\mathcal{R}a'$.

Le raisonnement est le même en partant de $a \xrightarrow{\epsilon(d)} a'$, nous avons donc la bisimulation pour les transitions continues.

– Considérons maintenant la transition discrète $s \xrightarrow{t} s'$ de $\mathcal{T}$. Puisque $s\mathcal{R}a$, le marquage de s et a est le même ainsi que les transitions sensibilisées. De plus, si t est franchissable à partir de a, alors a' et s' ont de façon évidente le même marquage M'.
Pour que t soit franchissable à partir de a, la garde correspondante dans l'automate doit être satisfaite. t est tirable à partir de s, donc $\nu_{\mathcal{T}}(t) \geq \alpha(t)$. $s\mathcal{R}a$ implique que $\exists x \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x)$. Donc, $\exists x \in X, \nu_{\mathcal{A}}(x) \geq \alpha(t)$. La garde associée à t, étant par construction de l'automate, $\nu_{\mathcal{A}}(x) \geq \alpha(t)$, elle est donc satisfaite.

Maintenant, considérons une transition t' sensibilisée par le marquage de s'. Deux situations sont possibles :

- t' n'est pas nouvellement sensibilisée par le tir de t, ce qui implique que t' était sensibilisée par le marquage de s . Alors, comme $s\mathcal{R}a$, $\exists x \in X, \nu_{\mathcal{A}}(x) = \nu_{\mathcal{T}}(t')$. Or $trans'(x) \neq \emptyset$, puisque nous avons au moins $t' \in trans'(x)$. Donc l'horloge x existe toujours dans la localité de a' et puisqu'aucun temps ne s'écoule lors d'une transition discrète, nous avons $\nu'_{\mathcal{A}}(x) = \nu'_{\mathcal{T}}(t')$.
- t' est nouvellement sensibilisée par le tir de t. Alors par construction, il existe une horloge (créée ou déjà existante et nulle) x dans a' et sa valuation est nulle. Nous avons donc $\nu'_{\mathcal{A}}(x) = \nu'_{\mathcal{T}}(t') = 0$.

Nous avons montré que s' et a' ont le même marquage et que $\forall t' \in enabled(M')$, $\exists x \in X, \nu'_{\mathcal{T}}(t') = \nu'_{\mathcal{A}}(x)$ donc $s'\mathcal{R}a'$.

Nous pouvons facilement montrer avec le même raisonnement que si $a \xrightarrow{t} a'$ pour $\Delta(\mathcal{T})$, alors $s \xrightarrow{t} s'$ conduit, pour $\mathcal{T}$, à un état s' tel que $s'\mathcal{R}a'$, d'où la bisimulation pour les transitions discrètes.

□

5.3.2 Finitude

Nous avons vu au chapitre précédent qu'un réseau de Petri T-temporel a un nombre fini de classes si et seulement si il est borné (Théorème 1). La preuve de [BD91] ne repose que sur les marquages et les domaines de tir, elle est donc tout fait applicable aux classes d'états étendues car χ et $trans(x), x \in \chi$ sont finis pour chaque classe. Nous avons donc le théorème suivant :

Théorème 5 *Un réseau de Petri T-temporel a un nombre fini de classes d'états étendues si et seulement si il est borné*

La problématique est donc la même que pour le calcul du graphe des classes classique. La bornitude étant indécidable, nous vérifions lors du calcul du graphe des classes étendues des conditions nécessaires de non-bornitude, afin de pouvoir l'arrêter le cas échéant. Ces conditions sont données par le théorème 6, qui est une extension triviale de celui de [BD91] donné au chapitre 4.

Théorème 6 *Un réseau de Petri T-temporel est borné s'il n'existe pas de paire de classes d'états étendues $C = (M, D, \chi, trans)$ et $C' = (M', D', \chi', trans')$, accessibles depuis la classe initiale telle que*

1. *C' est accessible depuis C ;*
2. *$M' > M$;*
3. *$[\![D']\!] = [\![D]\!]$;*
4. *$\forall p \in \{p \in P, M'(p) > M(p)\}, M(p) > \max_{t \in T} {}^\bullet t(p)$.*

5.3.3 Nombre d'horloges

Le nombre d'horloges d'un automate temporisé est un facteur très important dans l'efficacité des algorithmes de vérification. En effet, la complexité de ces algorithmes est, en

général, linéaire en le nombre de localités et exponentielle en le nombre d'horloges. Nous allons maintenant donner et prouver plusieurs propriétés démontrant le faible nombre d'horloges de l'automate temporisé des classes d'états.

Afin de générer aussi peu d'horloges que possible deux moyens principaux sont utilisés dans notre méthode. Premièrement, quand des transitions sont sensibilisées simultanément, leurs valuations restent égales le long de toute séquence de tir durant laquelle elles restent continûment sensibilisées. En effet, soit n transitions $t_1, \ldots, t_n$ sensibilisées simultanément à l'instant τ_0. Pour tout $k \in [\![1, n]\!]$, nous avons $\nu(t_k)(\tau_0) = 0$. Ensuite le temps s'écoule à la même vitesse pour toutes les transitions sensibilisées donc tant qu'aucune de ces n transitions n'est désensibilisée leurs valuations restent égales. Par conséquent, nous n'avons besoin que d'une horloge pour représenter les valuations de ces n transitions. De plus, nous n'avons besoin de cette horloge que tant que l'une au moins de ces n transitions n'a pas été désensibilisée.

Deuxièmement, nous voyons que la durée d'utilisation des horloges est bornée par le maximum des instants de tirs au plus tard des transitions qui lui sont associées. Par conséquent, les horloges dont toutes les transitions associées ont été désensibilisées sont réutilisées quand nous avons besoin d'une nouvelle horloge afin de représenter les valuations de transitions nouvellement sensibilisées. La politique est alors de choisir la première disponible, c'est-à-dire, l'horloge inutilisée qui a le plus petit indice.

Nous avons, de façon immédiate, le théorème suivant qui nous fournit un majorant du nombre d'horloges de l'automate temporisé des classes d'états :

Théorème 7 *L'automate temporisé des classes d'états $\Delta(\mathcal{T})$ d'un réseau de Petri T-temporel $\mathcal{T}$ possède un nombre d'horloges inférieur ou égal au nombre maximum de transitions sensibilisées par les marquages accessibles de $\mathcal{T}$.*

Pour des propriétés plus précises nous avons d'abord besoin de quelques définitions.

Si δ est la garde d'une arête de l'automate temporisé, alors nous notons $op(\delta) \subset X$ l'ensemble des horloges contraintes par δ. De la même façon, pour une localité l, $op(Inv(l)) \subset X$ est l'ensemble des horloges contraintes par l'invariant $Inv(l)$.

Nous définissons maintenant la notion d'*utilité* d'une horloge.

Définition 19 (TPN-utilité) $\textit{TPN-useful}(x) = \{l \in L | \exists (l, \delta, \alpha, R, l') \in E, x \in op(\delta) \cup op(Inv(l))\}$

Cette utilité caractérise les localités pour lesquelles une horloge particulière apparaît soit dans les gardes des transitions sortantes soit dans l'invariant.

À partir de cette définition, nous pouvons définir la notion d'*orthogonalité* de deux horloges et pour notre type particulier d'automate temporisé, nous donnons une définition équivalente de l'*activité* de Daws et Yovine [DY96].

Définition 20 (Orthogonalité de deux horloges) *x et y sont* orthogonales*, ce qui est noté $x \perp y$ si $\textit{TPN-useful}(x) \cap \textit{TPN-useful}(y) = \emptyset$*

Définition 21 (Activité) *Soit $Act(\mathcal{A})$ l'ensemble des horloges* actives *de l'automate temporisé $\mathcal{A}$.*

$$x \in Act(\mathcal{A}) \Leftrightarrow \textit{TPN-useful}(x) \neq \emptyset$$

Pour l'automate temporisé des classes d'états, il est clair que les horloges orthogonales pourraient être transformées en une horloge unique, réduisant ainsi le nombre total d'horloges. Cependant, nous avons le théorème suivant :

Théorème 8 *Il n'existe pas d'horloges orthogonales dans l'automate des classes d'états.*

Preuve. Soient $x_0, \ldots, x_n$ les horloges de l'automate temporisé des classes d'états. Quand x_k ($k \in [\![0, n]\!]$) a été créée, tous les indices inférieurs à k étaient utilisés puisque le nouvel indice est choisi comme le plus petit disponible. Par conséquent, x_k n'est orthogonal à aucun des $x_0, \ldots, x_{k-1}$. Puisque la relation d'orthogonalité est symétrique, cela prouve qu'il n'y a pas d'horloges orthogonales dans l'automate. □

Par ailleurs, d'après la définition précédente, l'automate temporisé des classes d'états n'a pas non plus d'horloges *inactives*, puisque dans chaque localité toutes les horloges apparaissent dans les invariants. Une autre propriété importante est que l'automate n'a pas non plus d'horloges *égales*.

Théorème 9 *Il n'existe pas de localité l de l'automate temporisé des classes d'états dans laquelle deux horloges x, y utilisées dans l ($l \in$ TPN-useful$(x) \cap$ TPN-useful(y)) sont égales.*

Preuve. Nous allons démontrer cela par induction sur l'automate. D'abord, la localité initiale n'a qu'une seule horloge, donc il n'y a rien à prouver pour cette localité.

Ensuite, générons une nouvelle localité. Si nous avons deux horloges égales x_i et x_j, alors il existe une localité précédemment générée dans lequel x_i a été créée. À l'entrée de cette localité, nous avions $\nu(x_i) = 0$ et puisque le temps s'écoule de la même façon pour toutes les horloges, nous avions également $\nu(x_j) = 0$. Mais une nouvelle horloge n'est pas créée s'il existe une horloge égale à 0, par définition. Par conséquent x_i et x_j ne peuvent pas être égales. □

En résumé, le nombre d'horloges de l'automate temporisé est *minimal* au sens des théorèmes 8 et 9, ce qui implique qu'il ne peut pas être réduit par les méthodes classiques de réduction d'horloge comme [DY96]. Cependant, si une transition est sensibilisée mais jamais tirée, cela peut entraîner la création d'une horloge qui ne sera jamais *réellement* utile. En effet, si la transition n'est jamais tirée, l'horloge correspondante peut n'apparaître dans aucune garde, et la partie des invariants due à cette transition est forcément redondante (l'horloge n'est cependant pas *inactive* au sens de la définition 21 car elle apparaît dans au moins un invariant). Cette situation ne peut pas être détectée à la volée par l'algorithme mais peut l'être par une analyse statique ultérieure. Cependant ce cas est assez rare, d'autant plus qu'il faut pour qu'une horloge inutile soit créée que cette transition soit sensibilisée seule (ou avec d'autres transitions jamais tirées) et même dans ce cas, l'horloge créée peut être réutilisée (ou avoir déjà été utilisée) ailleurs dans l'automate.

Nous illustrons l'efficacité de notre méthode en termes de nombre d'horloges en la comparant aux approches de [BST98], [SY96] et [Sav01]. Les résultats sont donnés dans le tableau 5.1. À notre connaissance, aucun outil n'est disponible qui implémente les deux premières méthodes aussi les chiffres donnés correspondent à des résultats théoriques. Par ailleurs, Sava ne donne pas d'implémentation de sa méthode et certains points ne sont pas résolus, aussi nous avons utilisé les solutions et l'implémentation proposées par [GRR05].

La première colonne présente les résultats de l'approche de [BST98]. Ils correspondent à une horloge par arc entrant de chaque transition. La traduction de [SY96] donne une horloge par place du réseau. Cependant, le modèle considéré n'est pas exactement les réseaux de Petri T-temporels aussi nous avons extrapolé ces résultats, qui sont donnés dans la deuxième colonne.

Les deux premières méthodes sont limitées à des réseaux saufs. C'est pourquoi certains résultats manquent dans les deux premières colonnes ; ils correspondent aux exemples pour

lesquels le réseau sous-jacent est non borné. L'algorithme de [Sav01] n'est pas sujet à ces restrictions et donne une horloge par transitions (colonne 3).

Ces trois algorithmes ne sont pas très efficaces en ce qui concerne le nombre d'horloges qu'ils produisent. Aussi nous avons appliqué l'algorithme hors-ligne de réduction du nombre d'horloges de [DY96] sur les résultats de l'algorithme de Sava, comme proposé dans [GRR05]. La réduction obtenue est très bonne (colonne 4). Mais les résultats obtenus par notre méthode, donnés dans la dernière colonne, restent meilleurs ; nous obtenons toujours autant ou moins d'horloges que toutes les autres approches.

	BST98	SY96	Sav01	Sav01 + DY96	Romeo
Exemple 1	NA	NA	12	8	6
Exemple 2	13	12	10	3	3
Exemple 3	14	14	14	2	2
Exemple 4	24	24	22	2	2
Exemple 5	31	29	23	3	2
Exemple 6	10	5	10	1	1
Exemple 7	NA	NA	20	11	7
Exemple 8	NA	NA	21	11	7
Exemple 9	NA	NA	15	3	3
Exemple 10	20	31	13	3	3
Exemple 11	12	20	9	4	4
Exemple 12	16	12	13	4	4
Exemple 13	20	16	17	4	4
Exemple 14	16	20	16	4	4
Exemple 15	NA	NA	31	4	2
Exemple 16	NA	NA	17	8	2
Exemple 17	NA	NA	13	9	4
Exemple 18	NA	NA	14	10	4
Exemple 19	NA	NA	20	13	3
Exemple 20	NA	NA	16	2	2

Tab. 5.1 – Nombre d'horloges

5.3.4 Taille

L'automate des classes d'états est en général plus petit, au sens du nombre d'états discrets et du nombre de transitions, que le graphe des classes d'états classique. Cela est illustré par le tableau 5.2 qui compare les deux approches sur les exemples utilisés à la sous-section 5.3.3. Étant donné que les opérations supplémentaires requises pour le calcul des horloges sont de faible complexité[4], cela a pour effet de rendre le calcul de l'automate des classes d'états bien plus rapide, en général, que celui du graphe des classes d'états.

Notons, qu'il existe des cas, où l'automate temporisé des classes d'états est plus gros que le graphe des classes d'états. Considérons par exemple, le réseau de la figure 5.7.

[4]Pseudo linéaire en le nombre d'horloges

	Locations (TA)	Transitions (TA)	Nodes (Graph)	Transitions (Graph)
Exemple 1	123	258	355	661
Exemple 2	33	47	59	79
Exemple 3	16	16	16	16
Exemple 4	23	24	23	24
Exemple 5	48	69	159	206
Exemple 6	5	10	5	10
Exemple 7	1169	4089	14418	46079
Exemple 8	1294	4358	13557	41249
Exemple 9	58	135	252	548
Exemple 10	39	68	126	222
Exemple 11	50	123	138	330
Exemple 12	131	351	4256	8977
Exemple 13	355	878	24401	50876
Exemple 14	1048	3002	22016	60967
Exemple 15	1088	5245	1098	5260
Exemple 16	76	199	200	353
Exemple 17	735	2263	1403	3508
Exemple 18	1871	6322	2831	8386
Exemple 19	11490	50168	14086	56929
Exemple 20	14	20	16	22

TAB. 5.2 – Nombre de nœuds et de transitions

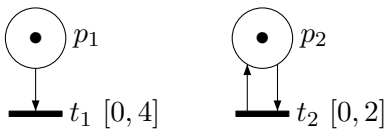

FIG. 5.7 – Un réseau de Petri T-temporel

La classe étendue initiale est :

$$C_0 = \begin{cases} \{p_1, p_2\}, \\ \begin{cases} 0 \leq \theta_1 \leq 4, \\ 0 \leq \theta_2 \leq 2, \end{cases} \\ x_0 : t_1, t_2 \end{cases}$$

Après le tir de t_2, nous obtenons la classe étendue suivante :

$$C_1 = \begin{cases} \{p_1, p_2\}, \\ \begin{cases} 0 \leq \theta_1 \leq 4, \\ 0 \leq \theta_2 \leq 2, \end{cases} \\ x_0 : t_1, \\ x_1 : t_2 \end{cases}$$

Les marquages et domaines de C_0 et C_1 sont les mêmes. Donc nous avons convergence pour la méthode du graphe des classes d'états, ce qui conduit à un graphe de 2 nœuds et 3 transitions. Cependant nous voyons que C_0 et C_1 ne sont pas *clock*-similaires. Par conséquent, l'automate temporisé des classes d'états compte 3 localités et 5 transitions.

5.4 Vérification avec l'automate temporisé des classes d'états

Grâce à l'automate temporisé des classes d'états, nous pouvons faire de la vérification efficace sur un réseau de Petri T-temporel avec UPPAAL ou KRONOS par exemple. Nous proposons, dans cette section, un cadre formel pour exprimer les propriétés directement en TCTL sur le réseau de Petri T-temporel puis pour les traduire sur l'automate temporisé des classes d'états afin de faire la vérification. Cela est possible grâce à la bisimulation qui nous assure l'égalité des valeurs de vérité des propriétés sur le réseau de Petri et sur l'automate, et efficace grâce au faible nombre d'horloges de l'automate.

Puisque TCTL est décidable pour les automates temporisés, la bisimulation que nous avons prouvée démontre le théorème suivant :

Théorème 10 *TCTL est décidable pour les réseaux de Petri T-temporels bornés.*

Cela justifie notamment le choix de TCTL par rapport, par exemple, à TLTL (*Timed Linear Temporal Logic*) qui elle n'est pas décidable pour les automates temporisés [AH94].

Nous donnons maintenant une définition de TCTL [ACD93, HNSY94] pour les réseaux de Petri T-temporels. La seule différence avec les versions de [ACD93, HNSY94] est que les propriétés atomiques habituellement associées aux états sont des propriétés de marquage.

5.4.1 TCTL pour les réseaux de Petri T-temporels

Soit un réseau de Petri T-temporel $\mathcal{T}$. Notons $[\![\mathcal{T}]\!]$ l'ensemble des séquences de tir de transitions dans l'espace d'états.

Définition 22 (TPN-TCTL) *Soit un réseau de Petri T-temporel* $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, \alpha, \beta)$ *avec* $P = \{p_1, \cdots, p_m\}$ *et* $T = \{t_1, \cdots, t_n\}$. *La logique temporelle* TPN-TCTL *est définie inductivement par :*

$$\textit{TPN-TCTL} ::= M(p_i) \bowtie V \mid t_i - t_j \leq d \mid M_i - M_j \leq d \mid \mathbf{faux} \mid \neg\varphi \mid \varphi \rightarrow \psi \mid \varphi \exists\mathcal{U}\bowtie c\psi \mid \varphi \forall\mathcal{U}\bowtie c\psi$$

où **M** *et* **faux** *sont des mots-clés,* $t_i, t_j \in T$, $M_i, M_j \in \mathbb{N}^p$, $\varphi, \psi \in$ *TPN-TCTL,* $p_i \in P$, $c, d, V \in \mathbb{N}$ *et* $\bowtie \in \{<, \leq, =, >, \geq\}$.

Intuitivement, $M(p_i) \bowtie V$ signifie que le marquage actuel de la place p_i est en relation $\bowtie$ avec V. $t_i - t_j \leq d$ signifie qu'à partir de l'état courant, tout tir de t_i est suivi d'un tir de t_j moins de d unités de temps plus tard. De la même façon, $M_i - M_j \leq d$ signifie qu'à partir l'état courant, toute occurrence du marquage M_i est suivie d'une occurrence du marquage M_j moins de d unités de temps plus tard. Les autres opérateurs ont leur signification habituelle. Nous utilisons également les raccourcis habituels $\mathbf{true} = \neg\mathbf{false}$, $\exists\Diamond_{\bowtie c}\phi = \mathbf{true}\exists\mathcal{U}_{\bowtie c}\phi$ et $\forall\Box_{\bowtie c} = \neg\exists\Diamond_{\bowtie c}\neg\phi$.

La sémantique de TPN-TCTL est définie sur les systèmes de transitions temporisés. Soit un réseau de Petri T-temporel $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, \alpha, \beta)$ et $S_\mathcal{T} = (Q, q_0, \rightarrow)$ la sémantique

$$
\begin{array}{lll}
(M,\nu) \models M(p_i) \bowtie V & \text{ssi} & M(p_i) \bowtie V \\
(M,\nu) \models t_i - t_j \leq d & \text{ssi} & \forall \sigma = (s_0,\nu_0) \overset{a_1,d_1}{\rightarrow} (s_1,\nu_1) \cdots \overset{a_n,d_n}{\rightarrow} (s_n,\nu_n) \in [\![\mathcal{T}]\!] \\
 & & \text{t.q. } (s_0,\nu_0) = (M,\nu), \text{nous avons} \\
 & & \forall k,l \in [\![1,n]\!], \text{ t.q. } \begin{cases} a_k = t_i, a_l = t_j, k < l, \\ \forall m, k < m < l, a_m \neq t_i, \end{cases} \\
 & & \sum_{n=k+1}^{n=l} d_n \leq d \\
(M,\nu) \models M_i - M_j \leq d & \text{ssi} & \forall \sigma = (s_0,\nu_0) \overset{a_1,d_1}{\rightarrow} (s_1,\nu_1) \cdots \overset{a_n,d_n}{\rightarrow} (s_n,\nu_n) \in [\![\mathcal{T}]\!] \\
 & & \text{t.q. } (s_0,\nu_0) = (M,\nu), \text{nous avons} \\
 & & \forall k,l \in [\![1,n]\!] \text{ t.q. } \begin{cases} s_k = M_i, s_l = M_j, k < l, \\ \forall m, k < m < l, s_m \neq M_i, \end{cases} \\
 & & \sum_{n=k+1}^{n=l} d_n \leq d \\
(M,\nu) \not\models \textbf{faux} & & \\
(M,\nu) \models \neg\varphi & \text{ssi} & (M,\nu) \not\models \varphi \\
(M,\nu) \models \varphi \rightarrow \psi & \text{ssi} & (M,\nu) \models \varphi \text{ implique } (M,\nu) \models \psi \\
(M,\nu) \models \varphi \exists \mathcal{U} \bowtie c \psi & \text{ssi} & \exists \sigma = (s_0,\nu_0) \overset{a_1,d_1}{\rightarrow} (s_1,\nu_1) \cdots \overset{a_n,d_n}{\rightarrow} (s_n,\nu_n) \in [\![\mathcal{T}]\!] \\
 & & \text{t.q. } \begin{cases} (s_0,\nu_0) = (M,\nu) \\ \forall i \in [\![1,n]\!], \forall d \in [0,d_i[, (s_i,\nu_i + d) \models \varphi \ et \\ \left(\sum_{i=1}^{n} d_i\right) \bowtie c \ et \ (s_n,\nu_n) \models \psi \end{cases} \\
(M,\nu) \models \varphi \forall \mathcal{U} \bowtie c \psi & \text{ssi} & \forall \sigma = (s_0,\nu_0) \overset{a_1,d_1}{\rightarrow} (s_1,\nu_1) \cdots \overset{a_n,d_n}{\rightarrow} (s_n,\nu_n) \in [\![\mathcal{T}]\!] \\
 & & \text{t.q. } (s_0,\nu_0) = (M,\nu), \text{ nous avons} \\
 & & \forall i \in [\![1,n]\!], \forall d \in [0,d_i[, (s_i,\nu_i + d) \models \varphi \ et \left(\sum_{i=1}^{n} d_i\right) \bowtie c \ et \ (s_n,\nu_n) \models \psi
\end{array}
$$

TAB. 5.3 – Sémantique de TPN-TCTL

de $\mathcal{T}$. La valeur de vérité de la formule φ de TPN-TCTL pour un état (M, ν) est donnée par la table 5.3.

Le réseau de Petri T-temporel $\mathcal{T}$ satisfait la formule φ de TPN-TCTL, ce que nous notons $\mathcal{T} \models \varphi$, ssi l'état initial de $S_{\mathcal{T}}$ satisfait φ c'est-à-dire $(M_0, \nu_0) \models \varphi$.

5.4.2 Vérification pour TPN-TCTL

Pour toutes les propriétés, exceptées $t_i - t_j \leq d$ et $M_i - M_j \leq d$, la formule exprimée sur le réseau de Petri T-temporel est traduite de façon triviale sur l'automate temporisé des classes d'états. Le traitement étant différent pour ces deux types de propriétés, nous définissons $TPN - TCTL^*$ comme $TPN - TCTL$ privé de ces deux types de propriétés :

$$TPN - TCTL^* ::= M(p_i) \bowtie V \,|\, \textbf{faux} \,|\, \neg\varphi \,|\, \varphi \rightarrow \psi \,|\, \varphi \exists\mathcal{U} \bowtie c\psi \,|\, \varphi \forall\mathcal{U} \bowtie c\psi$$

Vérification pour TPN-TCTL*

Puisque $\mathcal{T}$ et $\Delta(\mathcal{T})$ sont temporellement bisimilaires, nous avons le théorème suivant :

Théorème 11 (Vérification de TPN-TCTL*) *Soit $\mathcal{T}$ un réseau de Petri T-temporel et $\Delta(\mathcal{T})$ son automate temporisé des classes d'états.*

$$\forall \varphi \in \textit{TPN-TCTL*}, \mathcal{T} \models \varphi \Leftrightarrow \Delta(\mathcal{T}) \models \varphi$$

Pratiquement, avec Uppaal par exemple, le marquage est ajouté à l'automate sous la forme d'une variable M de type *tableau d'entiers*. L'automate met à jour M de telle façon que $M[i]$ est le nombre de jetons dans la place p_i du réseau. Par exemple, si nous voulons vérifier $\mathcal{T} \models \forall\Box_{\leq 3}(M(p_1) \geq 1 \wedge M(p_2) \leq 2)$, ce qui signifie que tous les états atteints dans les prochaines 3 unités de temps auront un marquage tel que p_1 a plus d'un jeton et p_2 moins de 2. Il est équivalent de vérifier $\forall\Box_{\leq 3}((M[1] \geq 1 \wedge M[2] \leq 2))$ sur l'automate des classes d'états.

Vérification pour TPN-TCTL

Pour les propriétés $t_i - t_j \leq d$ et $M_i - M_j \leq d$, utiliser directement les horloges de l'automate temporisé des classes peut s'avérer hasardeux. En effet, du fait de la réutilisation de ces horloges, il faut s'assurer que les propriétés ne sont vérifiées que pour des situations où les horloges représentent bien les valuations des transitions auxquelles nous souhaitons faire référence. Cela peut être fait en ajoutant des contraintes de marquage dans la propriété. Cependant, cela peut devenir assez fastidieux lorsque l'automate temporisé ne représente pas un cas d'école. Aussi pour des raisons de généricité et de facilité de modélisation de la propriété, nous proposons une approche par observateurs assez différente de celle de [TSLT97]. Nous pouvons ne jamais avoir à regarder les horloges de l'automate temporisé des classes d'états en le synchronisant avec de petits motifs d'automates temporisés très génériques, qui observent les tirs de transitions ou les occurrences de marquages. Ces observateurs fournissent des horloges additionnelles qui permettent de spécifier la propriété.

La synchronisation entre l'automate des classes d'états et les observateurs utilise la notion classique de composition basée sur une *fonction de synchronisation* à la Arnold-Nivat [Arn94].

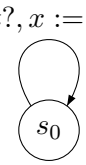

FIG. 5.8 – Automate observateur des tirs de transition

Définition 23 (Fonction de synchronisation) *Soit $\mathcal{A}_1, \ldots, \mathcal{A}_n$ n automates temporisés avec $\mathcal{A}_i = (L_i, l_{i,0}, X_i, A, E_i, Inv_i)$. Une* fonction de synchronisation *f est une fonction partielle $(A \cup \{\bullet\})^n \to A$ où $\bullet$ est un symbole spécial utilisé quand un automate n'est pas impliqué dans une transition du système global. Remarquons que f ainsi définie est une fonction de synchronisation avec renommage.*

Nous notons $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$ la composition parallèle des $\mathcal{A}_i$ par rapport à f. Les configurations de $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$ sont des couples $(\overline{l}, \nu)$ avec $\overline{l} = (l_1, \ldots, l_n) \in L_1 \times \ldots \times L_n$ et $\nu = \nu_1 \cdots \nu_n$ avec[5] $\nu_i \in (\mathbb{R}^+)^{X_i}$ (nous supposons que les ensembles d'horloges C_i sont disjoints). Alors la sémantique d'un produit synchronisé d'automates temporisés est également un système de transitions temporisé formalisé par la définition suivante :

Définition 24 (Sémantique d'un produit d'automates temporisés) *Soit $\mathcal{A}_1, \ldots, \mathcal{A}_n$ n automates temporisés avec $\mathcal{A}_i = (L_i, l_{i,0}, X_i, A, E_i, Inv_i)$, et f une fonction partielle de synchronisation $(A \cup \{\bullet\})^n \hookrightarrow A$. La sémantique du produit d'automates temporisés $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$ est un système de transitions temporisé $S = (Q, q_0, \to)$ avec $Q = L_1 \times \ldots \times L_n \times (\mathbb{R}^+)^C$, q_0 est l'état initial $((l_{1,0}, \ldots, l_{n,0}), \overline{0})$ et $\to$ est définie par :*

- $(\overline{l}, \nu) \xrightarrow{b} (\overline{l'}, \nu')$ *ssi il existe* $(a_1, \ldots, a_n) \in (A \cup \{\bullet\})^n$ *t.q.* $f(a_1, \ldots, a_n) = b$ *et pour tout* i *:*
  - *si* $a_i = \bullet$, *alors* $l'_i = l_i$ *et* $\nu'_i = \nu_i$,
  - *si* $a_i \in A$, *alors* $(l_i, \nu_i) \xrightarrow{a_i} (l'_i, v'_i)$.
- $(\overline{l}, \nu) \xrightarrow{t} (\overline{l}, \nu')$ *ssi* $\forall i \in [\![1, n]\!]$, $(l_i, \nu_i) \xrightarrow{t} (l_i, \nu'_i)$

De façon équivalente, nous pourrions définir le produit de n automates temporisés syntaxiquement en construisant un nouvel automate temporisé [LPY95] à partir des n initiaux. Dans la suite, nous considérons le produit $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$ comme un automate temporisé dont la sémantique est temporellement bisimilaire à celle de la définition 24.

Propriétés sur les instants de tir des transitions. Si nous voulons faire référence à l'instant de tir de la transition t dans une propriété, nous synchronisons l'automate temporisé des classes d'états avec l'automate temporisé de la figure 5.8.

L'observateur (Figure 5.8) a une localité, une transition et une horloge. La transition est synchronisée avec le tir de la transition t et l'horloge x est donc remise à zéro à chaque tir de t.

La propriété $t_i - t_j \leq d$ du réseau de Petri T-temporel $\mathcal{T}$ est alors vérifiée en ajoutant deux automates observateurs O_i et O_j fournissant respectivement les horloges x_i et x_j. Il suffit alors de vérifier la propriété $x_i - x_j \leq d$ sur le produit $(\Delta(\mathcal{T}) | O_i | O_j)_f$, avec la fonction de synchronisation à trois paramètres f définie par :

[5] ν_i est la restriction de ν à X_i.

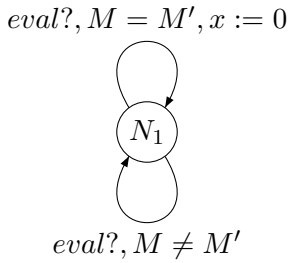

FIG. 5.9 – Automate observateur pour les occurrences de marquages

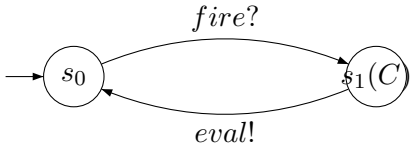

FIG. 5.10 – Automate superviseur pour les observateurs de marquage

– $f(t_i, t_i?, \bullet) = t_i$
– $f(t_j, \bullet, t_j?) = t_j$

Avec ces notations, nous avons le théorème suivant :

Théorème 12 (Vérification de $t_i - t_j \leq d$) *Soit $\mathcal{T}$ un réseau de Petri T-temporel et $\Delta(\mathcal{T})$ son automate temporisé des classes d'états.*

$$\mathcal{T} \models t_i - t_j \leq d \Leftrightarrow (\Delta(\mathcal{T})|O_i|O_j)_f \models x_i - x_j \leq d$$

Remarquons que tant que ni t_i ni t_j n'est tirée, les deux horloges x_i et x_j des automates observateurs correspondants sont égales donc $x_i - x_j = 0$ ce qui implique la propriété est vraie. Ensuite, si le premier tir de t_i intervient avant le premier tir de t_j alors $x_i - x_j \leq 0$ et par conséquent la propriété est toujours vraie.

Propriétés sur les instants d'occurrence des marquages. De façon similaire, si nous voulons faire référence à l'instant d'occurrence du marquage M, nous synchronisons l'automate temporisé des classes d'états avec l'automate temporisé de la figure 5.9. En effet, bien que les tirs de transitions et les occurrences de marquages sont fortement liés, il est bien plus pratique de faire référence à un marquage directement qu'aux transitions qui y ont conduit.

Afin de se synchroniser sur les changements de marquages, nous avons également besoin d'un superviseur S (Figure 5.10). Ce dernier est synchronisé sur toutes les transitions du réseau de Petri T-temporel, et détecte ainsi tout changement potentiel de marquage. Il force alors tous les automates observateurs de marquage à évaluer si le marquage courant est celui qu'ils observent ou non. Les observateurs de marquage fonctionnent de la même façon que les observateurs de transitions : leur horloge est remise à zéro lors de toute occurrence du marquage observé.

La synchronisation entre le superviseur et les observateurs de marquage est faite avec une localité *urgente*, que l'automate doit quitter immédiatement[6], et un vecteur de synchronisation sur la transition sortante de cette place qui synchronise le superviseur avec *tous* les observateurs de marquage. Cependant, les vecteurs de synchronisation ne sont pas autorisés par UPPAAL par exemple. Nous pouvons alors simuler ce mécanisme avec un compteur pour se synchroniser avec tous les observateurs de marquage successivement. Dans ce cas, les synchronisations ne sont pas strictement simultanées mais se font en 0 unités de temps. C'est pourquoi les formules TCTL sur les occurrences de marquages doivent être évaluées uniquement pour les états où le superviseur est dans sa localité initiale, ce qui assure que toutes les synchronisations ont bien eu lieu.

La propriété $M_i - M_j \leq d$ du réseau de Petri T-temporel $\mathcal{T}$ est alors vérifiée en ajoutant les observateurs O_i et O_j qui fournissent respectivement les horloges x_i et x_j. La propriété $s_0 \Rightarrow x_i - x_j \leq d$ est alors vérifiée sur le produit $(\Delta(\mathcal{T})|S|O_i|O_j)_f$, où f est la fonction de synchronisation à 4 paramètres définie par :

- $\forall k \in [\![1, n]\!], f(t_k, fire?, \bullet, \bullet) = t_k$
- $f(\bullet, eval!, eval?, eval?) = eval!$

Avec ces notations, nous avons le théorème suivant :

Théorème 13 (Vérification de $M_i - M_j \leq d$) *Soit $\mathcal{T}$ un réseau de Petri T-temporel et $\Delta(\mathcal{T})$ son automate temporisé des classes d'états.*

$$\mathcal{T} \models M_i - M_j \leq d \Leftrightarrow (\Delta(\mathcal{T})|S|O_i|O_j)_f \models s_0 \Rightarrow x_i - x_j \leq d$$

Remarquons que ces observateurs sont génériques. Il n'y a pas besoin d'en changer pour chaque propriété. De plus, ils ne participent pas à l'explosion combinatoire du nombre d'états discrets car ils n'ont qu'une seule localité (deux pour le superviseur pour les propriétés sur les marquages). Ils ajoutent néanmoins chacun une horloge, réduisant donc légèrement le gain obtenu lors de la traduction.

Par ailleurs, nous avons vu qu'il n'y en a pas besoin pour les propriétés de $TPN-TCTL^*$. Pour $TPN - TCTL$, nous pouvons également souvent nous en passer ; par l'utilisation des *propositions* (qui peuvent être des marquages) avec KRONOS, par exemple. L'introduction de ces observateurs relève donc d'une volonté de généricité pour la méthode que nous proposons.

5.4.3 Exemple

Considérons le réseau de Petri T-temporel $\mathcal{T}$ de la figure 5.4 de la sous-section 5.2.4. Nous pouvons vérifier par exemple que la tâche *consommatrice* est bloquée sur le sémaphore pendant une durée maximale inférieure à 3 unités de temps. Cela se traduit par une propriété sur la durée maximale entre les tirs de T_2 et $semP$. Nous pouvons donc exprimer la propriété à vérifier par « $T_2 - semP \leq 3$ » et nous avons donc besoin des deux observateurs O_2 et O_{semP} de la figure 5.11. Ces observateurs nous fournissent deux nouvelles horloges x_2 et x_{semP}. La formule TCTL à vérifier sur le produit $(\Delta(\mathcal{T})|O_2|O_{semP})_f$ est alors $x_2 - x_{semP} \leq 3$.

[6]Nous pouvons modéliser une telle localité avec une horloge x remise à zéro à l'entrée dans la localité et un invariant $x \leq 0$.

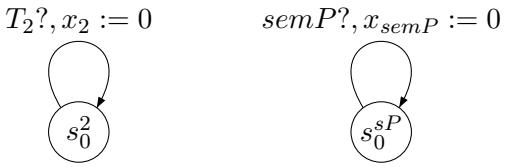

FIG. 5.11 – Observateurs O_2 et O_{semP} des transitions T_2 et $semP$

5.5 Conclusion

Dans ce chapitre, nous avons proposé une méthode pour calculer l'espace d'états d'un réseau de Petri T-temporel sous la forme d'un automate temporisé. Nous avons prouvé que cet automate et le réseau de Petri T-temporel sont temporellement bisimilaires. Nous pouvons donc dire que l'automate temporisé des classes d'états est une traduction du réseau de Petri T-temporel initial préservant sa sémantique comportementale. Cette bisimulation nous permet également d'affirmer que TCTL est décidable pour les réseaux de Petri T-temporel et nous avons donné un cadre formel pour l'expression de formules TCTL directement sur le réseau de Petri T-temporel et leur vérification au moyen de l'automate temporisé des classes d'états. Cette méthode de vérification est efficace pour deux raisons : d'une part l'automate temporisé est souvent calculé plus vite que le graphe des classes car il est, en général, plus petit et les opérations supplémentaires dues aux horloges sont peu coûteuses. D'autre part, le nombre d'horloges de l'automate temporisé des classes d'états est « minimal » dans le sens où il ne peut être réduit par l'algorithme de [DY96], ce qui assure une vérification efficace par les algorithmes de *model-checking*.

À la fin de cette partie, nous avons donc une méthode de vérification « de référence » pour l'analyse des réseaux de Petri T-temporels.

Dans la partie suivante, nous allons étendre le modèle réseau de Petri T-temporel pour prendre en compte le problème spécifique de l'ordonnancement. Puis nous étendrons également notre méthode d'analyse.

Troisième partie

Modèles à chronomètres

Chapitre 6

Réseaux de Petri T-temporels étendus à l'ordonnancement

Les réseaux de Petri T-temporels étendus à l'ordonnancement (*Scheduling*-TPN, *Scheduling Time Petri Nets* en anglais) ont été introduits par Olivier H. Roux et Anne-Marie Déplanche dans [RD02]. Ils étendent les réseaux de Petri T-temporels en incluant dans la sémantique du modèle le comportement des ordonnanceurs temps réel. Cela implique la suspension et la reprise de l'exécution des tâches, lors des préemptions. Le temps s'arrêtant pour les tâches préemptées, ce modèle s'appuie sur le concept de chronomètre (ou *stopwatch*), horloge pour laquelle le temps peut être arrêté.

Pour tout marquage M du réseau, nous définissons un sous-ensemble de M appelé *marquage actif*, et noté $Act(M)$, qui sensibilise exactement les transitions pour lesquelles le temps s'écoule. Ces transitions modélisent soit une tâche qui s'exécute (non préemptée), soit un service de l'exécutif. Dans le premier cas, une transition sensibilisée par M mais pas par $Act(M)$ représente une tâche *prête* mais qui ne s'exécute pas (non active).

Pour une politique d'ordonnancement à priorités fixes, la fonction Act peut être définie à partir de deux nouveaux attributs associés aux places du réseau. γ désigne l'identité du processeur auquel la place est affectée et ω la priorité, sur ce processeur, de la tâche dont le modèle contient cette place.

6.1 Exemple

Un exemple de *Scheduling*-TPN est présenté sur la figure 6.1 pour le cas particulier d'un ordonnancement à priorités fixes. Il représente deux tâches τ_1 et τ_2 en concurrence sur un même processeur, τ_2 étant la plus prioritaire. Le marquage initial est $\{P_1, P_3\}$. Cependant, comme les deux places sont affectées au même processeur, et que la priorité de P_3 est la plus grande, le marquage actif initial est $\{P_3\}$. Donc la première transition tirée sera T_3, ce qui traduit le fait que c'est la tâche τ_2 qui s'exécute initialement.

6.2 Définitions

Définition 25 (*Scheduling*-TPN) *Un réseau de Petri T-temporel étendu à l'ordonnancement est un 8-uplet* $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0, Act)$, *où*

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

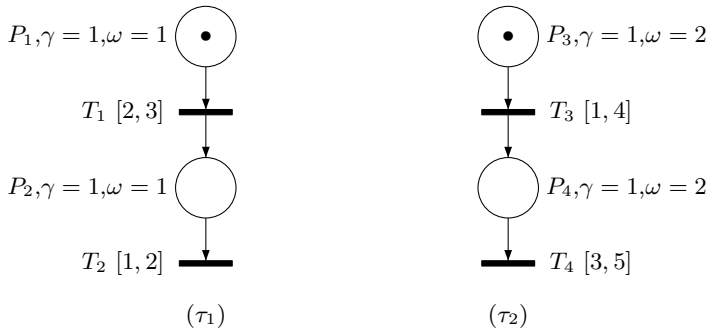

FIG. 6.1 – Deux tâches sur un même processeur ordonnancées par une politique à priorités fixes

- $T = \{t_1, t_2, \ldots, t_n\}$ *est un ensemble fini et non vide de* transitions $(T \cap P = \emptyset)$ *;*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *est la* fonction d'incidence amont *;*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *est la* fonction d'incidence aval *;*
- $M_0 \in \mathbb{N}^P$ *est le* marquage initial *du réseau ;*
- $\alpha \in (\mathbb{R}^+)^T$ *et* $\beta \in (\mathbb{R}^+ \cup \{\infty\})^T$ *sont les fonctions donnant pour chaque transition, respectivement son instant 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 *telle que* $\forall M \in \mathbb{N}^P, \forall p \in P, Act(M)(p) = M(p)$ *ou* $Act(M)(p) = 0$.

En plus des notions définies pour les réseaux de Petri T-temporels classiques, une transition t est dite *active* pour le marquage M, si elle est sensibilisée par le marquage $Act(M)$, c'est-à-dire $t \in enabled(Act(M))$.

Nous définissons la sémantique des réseaux de Petri T-temporels étendus à l'ordonnancement sous la forme d'un système de transitions temporisé.

Définition 26 (Sémantique d'un *Scheduling*-TPN) *La sémantique d'un réseau de Petri T-temporel étendu à l'ordonnancement $\mathcal{T}$ est définie sous la forme d'un système de transitions temporisé $S_{\mathcal{T}} = (Q, q_0, \rightarrow)$ tel que :*

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$ *;*
- $q_0 = (M_0, \overline{0})$ *;*
- $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ *est la relation de transition incluant des transitions continues et des transitions discrètes :*
  - *la relation de transition* continue *est définie* $\forall d \in \mathbb{R}^+$ *par :*

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

– *la relation de transition* discrète *est définie* $\forall t_i \in T$ *par :*

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

6.3 Marquage actif

La fonction donnant le marquage actif permet de définir la politique d'ordonnancement considérée pour le modèle. Nous pouvons constater que la façon dont est calculée cette fonction n'intervient pas dans la sémantique du modèle. Par conséquent, nous pouvons changer la politique d'ordonnancement sans rien changer au modèle d'autre que cette fonction.

Cependant, la façon dont intervient le marquage actif dans la sémantique impose des restrictions sur la politique d'ordonnancement. En effet, nous ne pouvons avoir de changement de marquage actif et donc d'état pour les tâches qu'à l'occurrence d'un événement discret modélisé par le tir d'une transition : par exemple l'arrivée d'une tâche ou sa terminaison. En particulier, des politiques d'ordonnancement du type de *Least Laxity* [MD78] ne sont pas représentables avec ce modèle.

Une des politiques d'ordonnancement les plus employées en pratique consiste à utiliser des priorités fixes. Ce cas a été étudié dans [RD02]. Le marquage actif est alors calculé à partir de deux nouveaux paramètres ajoutés aux places. Soit une place p, $\gamma(p)$ représente le processeur auquel la place est associée et $\omega(p)$ la priorité de cette place sur le processeur. $\gamma(p)$ peut prendre la valeur particulière ϕ lorsque p représente un service de l'exécutif comme une horloge, par exemple. Les places p telles que $\gamma(p) = \phi$ sont toujours actives, c'est-à-dire telles que $Act(M)(p) = M(p)$. Il est donc inutile de définir une priorité pour elle. Aussi dans les figures de ce document, les fonctions γ et ω sont généralement omises pour de telles places.

Les fonctions γ et ω étant données, Le calcul du marquage actif est le suivant :

1. déterminer l'ensemble $\mathcal{P}$ des places marquées qui sensibilisent au moins une transition ;
2. pour tout p de $\mathcal{P}$, telle que $\gamma(p) \neq \phi$, le marquage actif de p est son marquage « normal » si p est la place de plus haute priorité sur son processeur parmi les places de p_1, c'est-à-dire $Act(M)(p) = M(p)$ si $\omega(p) = \max\{\omega(p') | \, p' \in \mathcal{P}, \gamma(p') = \gamma(p)\}$. Sinon le marquage actif de p est nul.

Pour pouvoir calculer le marquage actif, nous imposons la restriction que toutes les places amont d'une même transition doivent avoir la même priorité. Sans cette restriction, nous pourrions avoir des relations de priorité circulaires empêchant le calcul du marquage actif. En pratique, la possibilité d'avoir plusieurs places amont d'une transition affectée à des processeurs est rarement utile : elle permet de modéliser la dépendance d'une action à plusieurs ressources préemptibles.

Dans la plupart des cas, il est préférable de respecter la règle suivante : toute transition doit avoir au plus une place amont p affectée à un « vrai » processeur, c'est-à-dire telle que $\gamma(p) \neq \phi$. Cela représente le fait que toute communication se fait par l'intermédiaire d'un service de l'exécutif.

Enfin, pour des raisons d'unicité du marquage actif, nous imposons que deux places affectées au même processeur, de même priorité et sensibilisant au moins une transition ne doivent pas être marquées simultanément. Cette restriction sera levée au chapitre 8.

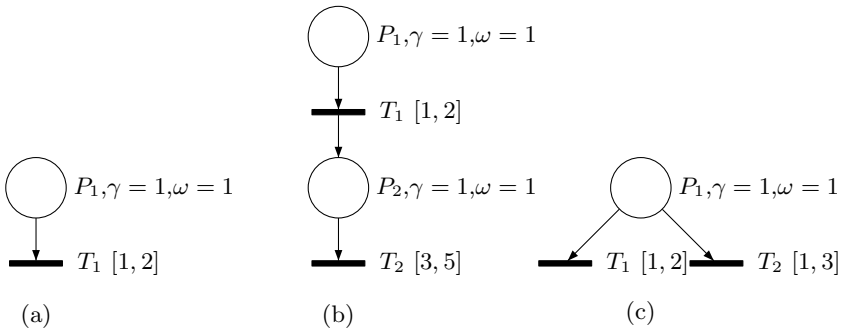

FIG. 6.2 – Modèle de tâches de base

6.4 Modélisation avec les *Scheduling*-TPN

Les réseaux de Petri T-temporels en général sont déjà beaucoup utilisés pour modéliser les systèmes temps réel et les protocoles. Le gain en expressivité des *Scheduling*-TPN tient dans leur capacité à modéliser les politiques d'ordonnancement et de synchronisation complexe fournies par les exécutifs temps réels tels que OSEK/VDX [gro01]. Nous allons maintenant illustrer sur quelques exemples les possibilités de modélisation des *Scheduling*-TPN dans le contexte des systèmes temps réel.

6.4.1 Modèle de tâches

Le motif de base que nous proposons pour représenter une tâche est donné sur la figure 6.2a. Ici quand la place P_1 est marquée, la tâche s'exécute et sa durée d'exécution est donnée par l'intervalle de temps de la transition T_1, c'est-à-dire entre 1 et 2 unités de temps. Ici la requête d'activation est implicite cependant il peut être intéressant, pour les besoins de la vérification, de la rendre explicite, sous la forme d'une transition $T_0[0, 0]$ en amont de P_1 et d'une place P_0 en amont de T_0. P_0 est alors marquée initialement (et pas P_1) et le tir de T_0 représente la requête d'activation de la tâche modélisée par P_1 et T_1.

Ces motifs de base peuvent être concaténés pour exprimer la séquentialité (figure 6.2b). Ils peuvent également être assemblés de façon à exprimer des comportements alternatifs (figure 6.2c). Pour la tâche modélisée, ce choix peut correspondre à un test déterministe, par exemple, du type `si... alors...` . Pour le modèle, le choix est indéterministe et les deux chemins seront envisagés lors de la vérification.

6.4.2 Activation des tâches

Les différents mécanismes d'activation des tâches sont facilement modélisables. La figure 6.3a montre une activation périodique. L'horloge associée à cette période est modélisée par la place P_x et la transition T_x dont l'intervalle de temps détermine la période.

La figure 6.3b montre une activation périodique décalée par rapport à l'origine des temps (*offset*). Enfin la figure 6.3c illustre une activation cyclique.

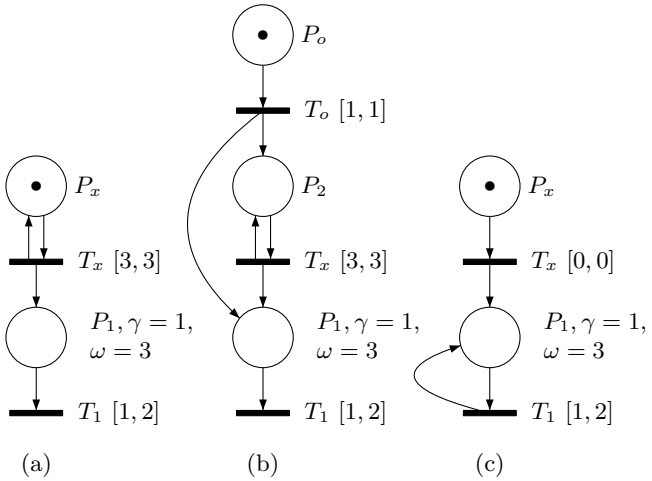

FIG. 6.3 – Mécanismes d'activation

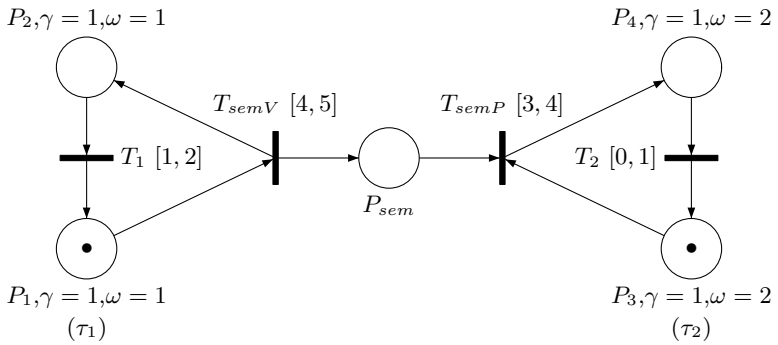

FIG. 6.4 – Deux tâches cycliques synchronisées par un sémaphore

6.4.3 Synchronisation de base

Les exécutifs temps réel fournissent, en général, un certain nombre de services de synchronisation des processus. En particulier, les sémaphores et les événements sont particulièrement utilisés. La figure 6.4 donne un modèle pour les sémaphores de Dijkstra. La tâche τ_2, qui est la tâche *consommatrice*, est la plus prioritaire. Nous imposons qu'elle attende l'autre tâche par un sémaphore. Le sémaphore est représenté par la place P_{sem} et sa valeur (quand elle est positive) est donnée par le nombre de jetons dans P_{sem}. Les actions classiques P et V sont modélisées respectivement par les transitions T_{semP} et T_{semV}. Tant qu'il n'y a pas de jeton dans la place P_{sem}, c'est-à-dire tant que la valeur du sémaphore est négative ou nulle, la tâche consommatrice est en attente car T_{semP} n'est pas tirable. Dès qu'un jeton est amené dans P_{sem} par le tir de T_{semV}[1], alors T_{semP} devient tirable et la tâche consommatrice peut faire l'action P.

La figure 6.5 fournit une modélisation possible d'une synchronisation par événement

[1]Nous pouvons remarquer que la tâche *productrice* ne peut être bloquée, le tir de T_{semV} étant toujours possible, ce qui est normal car l'action V n'est jamais bloquante.

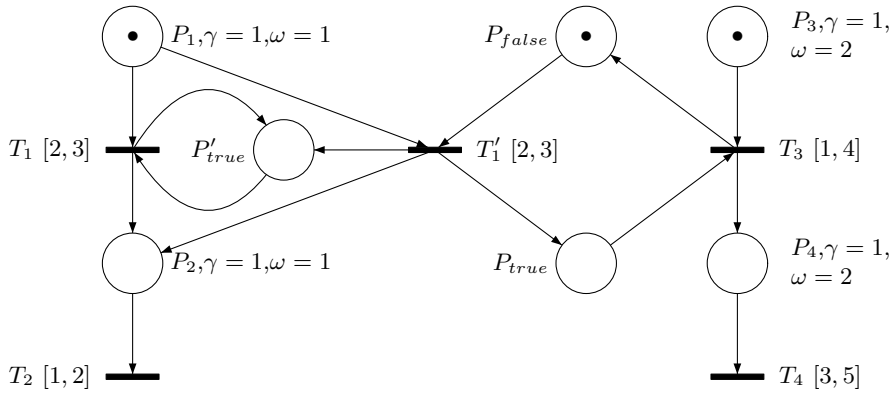

FIG. 6.5 – Synchronisation à l'aide d'un événement mémorisé

mémorisé telle que l'on peut en trouver dans OSEK/VDX, par exemple. La tâche de droite est la plus prioritaire, et comme pour l'exemple du sémaphore qui précède nous imposons qu'elle attende la tache moins prioritaire, cette fois grâce à un événement mémorisé. Un jeton dans P_{false} dénote l'absence de l'événement alors qu'un jeton dans P_{true} signifie que l'événement est présent. Nous pouvons remarquer qu'il ne peut y avoir un jeton dans ces deux places simultanément. La transition T_1 est dupliquée en une transition T'_1, de telle sorte que T_1 soit sensibilisée si et seulement si l'événement est présent et T'_1 est sensibilisée si et seulement si l'événement est absent. Cela permet de ne pas rendre la place P_{true} non bornée. P_{true} est également dupliquée en P'_{true} afin que les tirs de T_1 ne perturbent pas la sensibilisation de T_3. Au final, le premier tir de T'_1 déclenche l'événement et le tir de T_3 le consomme.

6.4.4 Accès aux ressources

Des mécanismes de synchronisation de plus haut niveau sont mis en œuvre pour l'accès de processus à des ressources partagées. Ces mécanismes relèvent de l'exclusion mutuelle. Une approche très utilisée pour assurer l'exclusion mutuelle entre des processus consiste à utiliser un sémaphore d'exclusion mutuelle. Nous pouvons le modéliser comme sur la figure 6.6. Le sémaphore est classiquement initialisé à 1 ce qui se traduit par un jeton dans P_{sem}. Chaque tâche entrant en section critique consomme le jeton pour le tir de la transition correspondant à cette entrée et le « rend » par le tir de la transition correspondant à la sortie de la section critique. Ainsi, tant qu'une tâche est en section critique, les transitions d'entrée en section critique des autres tâches ne sont pas sensibilisées ce qui assure l'exclusion mutuelle.

Cependant, il est bien connu que cette politique d'accès à une ressource critique peut entraîner des interblocages, aussi les exécutifs temps réel fournissent en général un mécanisme de plus haut niveau pour gérer l'accès aux ressources. Le plus couramment utilisé est le *protocole de la priorité plafond* (PCP, *Piority Ceiling Protocol* en anglais). C'est le protocole utilisé par OSEK/VDX, par exemple. Il peut être modélisé par le réseau de la figure 6.7. Sur cette figure, nous avons trois tâches périodiques en parallèle. Elles sont de plus en plus prioritaires de gauche à droite. Elles possèdent toutes les trois une section critique notée *sc*.

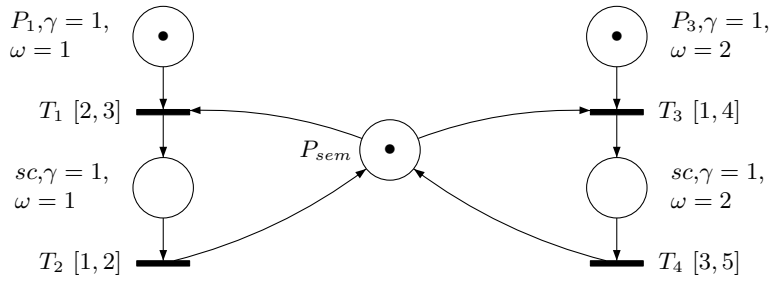

FIG. 6.6 – Sémaphore pour l'exclusion mutuelle

L'accès à la section critique est géré de manière analogue à celle décrite pour le sémaphore d'exclusion mutuelle. Mais, lorsque l'une des tâches entre en section critique, sa priorité est élevée à la priorité de la tâche la plus prioritaire pouvant accéder à cette section critique plus un. Elle devient donc la tâche la plus prioritaire et le restera jusqu'à sa sortie de la section critique.

6.4.5 Communication par réseau

Les réseaux de Petri T-temporels étendus à l'ordonnancement permettent de modéliser des systèmes multiprocesseurs. En particulier, ils permettent de modéliser facilement les communications entre les différents processeurs. La figure 6.8 propose une modélisation de haut niveau du protocole CAN. Dans cette modélisation, nous considérons le bus CAN comme une ressource critique pour laquelle l'accès est décidé en fonction des priorités découlant des identifiants de chaque message. Une modélisation bien plus précise avec des réseaux de Petri T-temporels peut-être trouvée dans [Jua99].

Ici, l'émetteur i ($i \in [\![0, n]\!]$) tente d'accéder au bus entre d_{i1} et d_{i2} unités de temps. L'accès est modélisé par les places *access* et se fait en 0 unités de temps par le tir de la transition t_{i1}. Si deux émetteurs tentent d'accéder au bus simultanément, c'est le message dont l'identifiant est le plus élevé qui est prioritaire grâce aux priorités sur les places *access*. L'autre émetteur tentera à nouveau d'émettre lorsque le bus sera libre, c'est-à-dire lorsqu'il y aura de nouveau un jeton dans la place $CANBus$.

6.5 Classification

Dans cette section, nous allons positionner le modèle réseaux de Petri T-temporel par rapport à d'autres extensions des réseaux de Petri. Pour cela, nous définissons et comparons les classes de modèles temporisés en termes d'acceptation de langage temporisé. Cela se traduit par la définition suivante de l'inclusion de classes de modèles :

Définition 27 (Sous-classe) *Une classe de modèles A est une* sous-classe *de la classe de modèles B si pour tout élément a de A, il existe un élément b de B tel que b accepte le même langage temporisé que a.*

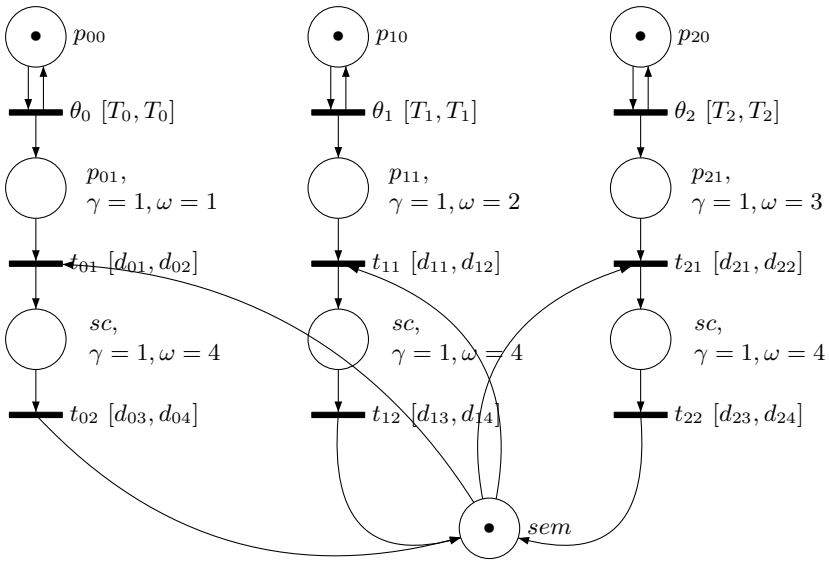

FIG. 6.7 – Protocole à Priorité Plafond (PCP)

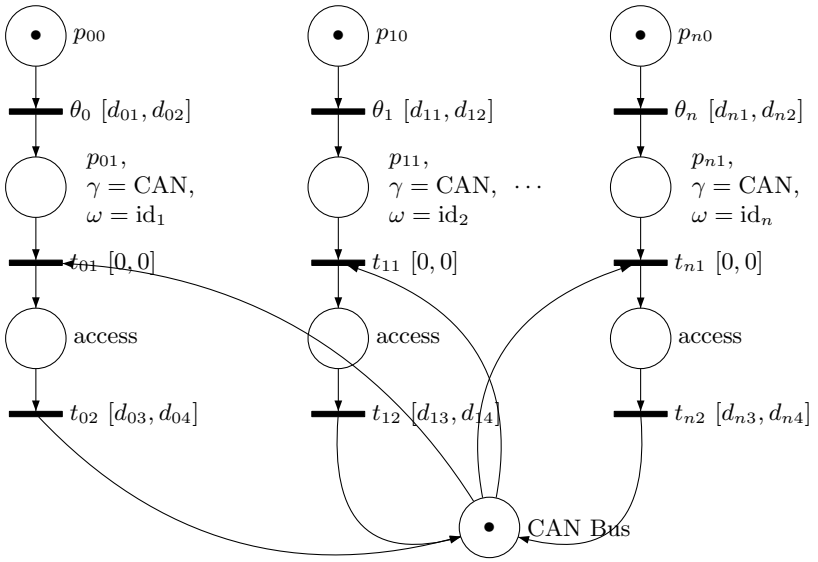

FIG. 6.8 – Accès à un bus CAN

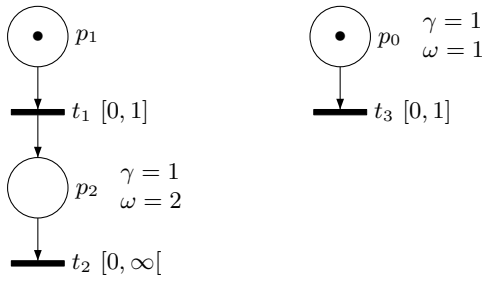

FIG. 6.9 – Un réseau de Petri T-temporel étendu à l'ordonnancement

Soit A incluse dans B. Si B est incluse dans A alors les classes A et B sont *égales* sinon A est une sous-classe *stricte* de B.

Nous allons comparer les réseaux de Petri T-temporels étendus à l'ordonnancement avec les classes de réseaux de Petri apparentées, à savoir les réseaux de Petri T-temporels et les réseaux de Petri T-temporels préemptifs (*Preemptive*-TPN) [BFSV03]. Pour cela, nous considérons le modèle *Scheduling*-TPN avec un marquage actif défini pour les politiques d'ordonnancement à priorités fixes de [RD02]. Avec d'autres définitions du marquage actif, la classe de modèles est susceptible de changer comme nous le verrons au chapitre 8.

Théorème 14 (Relation entre les TPN et les *Scheduling*-TPN) *Les réseaux de Petri T-temporels sont une sous-classe stricte des réseaux de Petri T-temporels étendus à l'ordonnancement.*

Preuve. De façon évidente, tout réseau de Petri T-temporel peut être représenté par un réseau de Petri T-temporel étendu à l'ordonnancement pour lequel toutes les places sont associées au processeur fictif ϕ.

Le réseau de la figure 6.9 reconnaît le langage temporisé $\forall n \in [0,1], \forall m \geq 0, (t_1, n)(t_2, m)(t_3, 1-n)$. Par contre, il n'existe pas de réseau de Petri T-temporel qui accepte ce langage temporisé. □

Les *Scheduling*-TPN et les *Preemptive*-TPN sont deux modèles très similaires. En particulier, le calcul de l'espace d'états peut être réalisé de la même façon. Néanmoins une traduction structurelle entre les deux modèles n'est pas triviale. Nous proposons donc la conjecture suivante :

Théorème 15 (Relation entre les *Preemptive*-TPN et les *Scheduling*-TPN) *Les réseaux de Petri T-temporels préemptifs et les réseaux de Petri T-temporels étendus à l'ordonnancement appartiennent à la même classe.*

Une collaboration est en cours avec Enrico Vicario et Luigi Sassoli de l'Université de Florence pour vérifier cette conjecture.

6.6 Indécidabilité

Nous venons de voir que les réseaux de Petri T-temporels étendus à l'ordonnancement sont une surclasse stricte des réseaux de Petri T-temporels. Puisque l'accessibilité de marquages et d'états, la bornitude et la vivacité sont indécidables pour les réseaux de Petri T-temporels classiques (voir le chapitre 4), ces problèmes sont indécidables pour l'extension à l'ordonnancement également.

Cependant, dans le cas des réseaux temporels, nous avons vu que la k-bornitude et, lorsque le réseau temporel est borné, l'accessibilité de marquages, l'accessibilité d'états et la vivacité sont des problèmes décidables. Il est donc légitime de se demander ce qu'il en est pour l'extension à l'ordonnancement.

Nous avons étudié ce problème avec Bernard Berthomieu et François Vernadat dans le cadre d'un modèle de réseaux de Petri T-temporels à chronomètres plus général [BLRV04]. En modélisant les machines à deux compteurs à l'aide de ce modèle nous avons montré que tous les problèmes « intéressants » cités au chapitre 4 sont indécidables, y compris, pour l'accessibilité et la vivacité, lorsque le réseau est borné. Une modélisation quasi-identique des machines à deux compteurs est réalisable avec les réseaux de Petri T-temporels étendus à l'ordonnancement et nous allons la présenter dans les paragraphes qui suivent.

6.6.1 Machine à deux compteurs

Définition 28 (Machine à deux compteurs) *Une* machine à deux compteurs *est un 6-uplet* $(Q, q_0, q_F, \mathcal{I}, c_1, c_2)$ *où*

- Q *est un ensemble fini d'états,*
- $q_0 \in Q$ *est l'état initial,*
- $q_F \in Q$ *est l'état final ou état d'arrêt,*
- c_1 *et* c_2 *sont des compteurs à valeurs entières positives ou nulles, et initialement nuls,*
- $\mathcal{I}$ *est un ensemble fini d'instructions, avec la forme et le sens suivant* ($i \in \{1, 2\}$) *:*
  - (p, dec_i, q) *: dans l'état* p*, décrémenter* c_i *de* 1 *et aller à l'état* q *;*
  - (p, inc_i, q) *: dans l'état* p*, incrémenter* c_i *de* 1 *and et aller à l'état* q *;*
  - $(p, test_i, q, r)$ *: dans l'état* p*, tester* c_i *; si* $c_i = 0$ *aller dans l'état* q*, sinon aller dans l'état* r*.*

La machine s'arrête lorsqu'elle atteint l'état q_F. Le problème de l'arrêt d'une machine à deux compteurs est indécidable [Min61].

6.6.2 Principes de la modélisation

La modélisation des machines à deux compteurs repose sur l'encodage de la valeur des compteurs comme la différence de phase $\phi_{e_1}^{e_2}(\tau)$ entre les dates d'occurrence de deux événements e_1 et e_2 périodiques de même période ρ à l'instant τ. Une valeur k du compteur c_i est alors encodée par une différence de phase $\phi_{tick}^{i.clock}$ entre le tir d'une transition de référence *tick* et le tir d'une transition *i.clock* représentant le compteur c_i : $\phi_{tick}^{i.clock}$ est la différence entre la date de tir de *i.clock* et la date de tir de *tick*. *tick* est invariablement tirée toutes les ρ unités de temps. Par contre, *i.clock* est en général tirée toutes les ρ unités de temps mais ce tir peut être ponctuellement retardé ou anticipé afin de changer la phase entre les deux événements et ainsi la valeur du compteur c_i. Ces changements de phase sont rendus possibles par la possibilité d'interdire temporairement l'écoulement du temps pour une transition dans ce modèle.

Ainsi nous modéliserons l'incrémentation du compteur c_i par une division par 2 de la valeur de la phase $\phi_{tick}^{i.clock}$ et une décrémentation du compteur par une multiplication de cette même valeur. Finalement nous avons donc $\phi_{tick}^{i.clock} = \rho/2^{c+1}$, c'est-à-dire qu'aux valeurs $0, 1, 2, \ldots$ du compteur correspondent les valeurs $\rho/2, \rho/4, \rho/8, \ldots$ de la phase. Cela nous permet d'encoder un espace discret infini dans un espace dense borné, comme cela a déjà été fait dans [HKPV98] et [Čer92]. Par la suite, l'événement de référence étant toujours *tick*, nous noterons $\phi_i(t)$ la phase de *i.clock* par rapport à *tick* à l'instant t et, par extension, $\phi_i(p)$ la phase de *i.clock* par rapport à *tick* à l'instant où la place p devient marquée.

La valeur exacte de ρ n'a pas d'importance tant qu'elle est supérieure à la phase maximale possible ; nous avons choisi $\rho = 8$. Par conséquent les valeurs $0, 1, 2, 3, \ldots$ d'un compteur correspondent aux différences de phase $4, 2, 1, \frac{1}{2}, \ldots$.

La modélisation est donnée sous la forme de motifs réseaux de Petri T-temporels étendus à l'ordonnancement. Ces motifs sont ensuite assemblés pour modéliser les instructions de la machine.

Afin de prouver que la modélisation est correcte, nous donnons, pour chaque motif un lemme décrivant son comportement, sous la forme $P \rightsquigarrow Q$, ce qui est équivalent à $P \Rightarrow \forall\Box\exists\Diamond Q$ où $\forall\Box$ et $\exists\Diamond$ sont les modalités CTL classiques. Typiquement nous aurons $in \wedge \phi_r^{in} = k \rightsquigarrow out \wedge \phi_r^{out} = k'$, ce qui signifie que si la place *in* est marquée et que la phase de l'événement r par rapport à $Tick$ est égale à k au moment où *in* est marquée, alors il sera toujours possible d'avoir la place *out* marquée dans le futur et la phase de r par rapport à $Tick$ sera égale à k' quand *out* sera marquée.

6.6.3 Événements de base et initialisation

Les événements de base que nous utilisons sont donc des tirs périodiques de transition, avec la période ρ. Nous en avons une pour chaque compteur, 1.*clock* et 2.*clock*, et une pour l'événement de référence *tick*. La période 8 de ces tirs de transition est découpée en $1 + 7$ afin de créer des points de synchronisation explicites.

Nous utiliserons par ailleurs deux autres événements, 3.*clock* et *guess* de façon temporaire, pour l'instruction d'incrémentation.

Enfin, nous initialisons chaque compteur à 0 en donnant des phases initiales de 4 en utilisant le motif Init. Les places *in* et *out* ne font pas partie du motif et matérialisent son contexte d'utilisation.

Les motifs décrits dans ce paragraphe sont donnés sur la figure 6.10.

Par ailleurs, nous avons de façon triviale

Lemme 1 (Initialisation)
$(N1)$ $in \rightsquigarrow out \wedge \phi_1(out) = \phi_2(out) = 4$

6.6.4 Instruction de test à zéro

Pour tester si la valeur du compteur c_i est nulle, nous devons tester si la valeur de la phase est égale à 4.

Lemme 2 (Test à zéro)
$(T1)$ $in \wedge \phi_i(in) = k \wedge k \leq 2 \rightsquigarrow outnz \wedge \phi_i(outnz) = k$
$(T2)$ $in \wedge \phi_i(in) = 4 \rightsquigarrow outz \wedge \phi_i(outz) = 4$

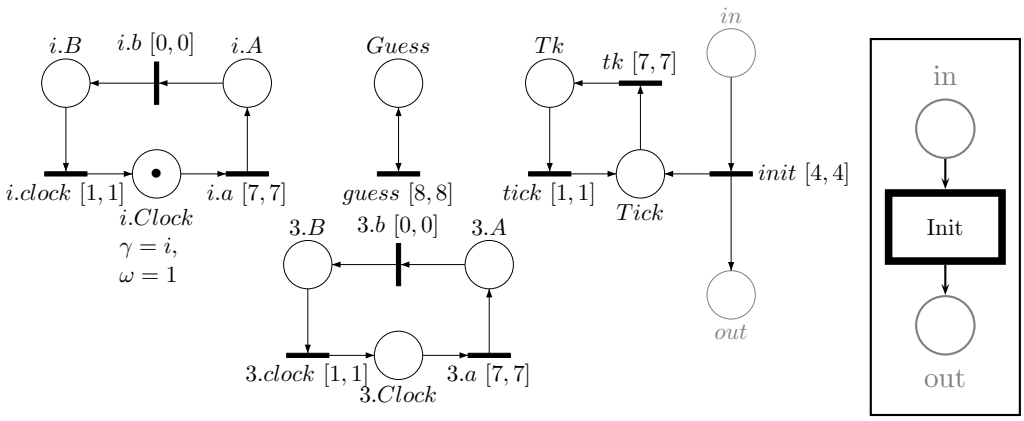

FIG. 6.10 – Événements périodiques de base et motif d'initialisation ($i \in \{1, 2\}$)

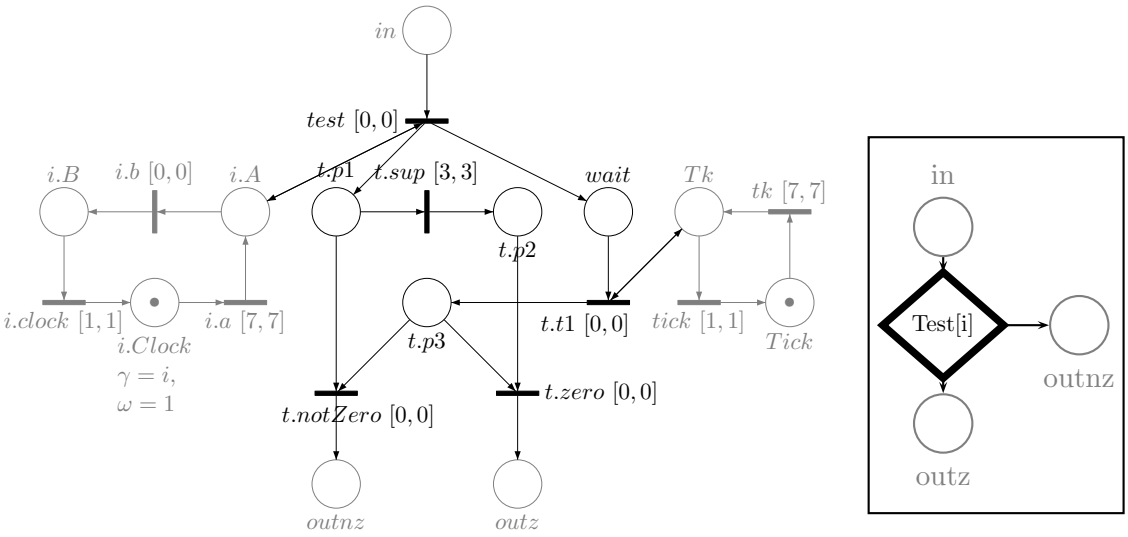

FIG. 6.11 – Motif de test à zéro du compteur c_i ($i \in \{1, 2\}$)

Preuve. Supposons que in soit marquée et que $\phi_i(in) = k$ avec $0 < k \leq 4$. Cela signifie que $tick$ est tirée k unités de temps après $i.clock$. Donc tk est tirée k unités de temps après $i.a$. Donc $test$ est tirée k unités de temps avant tk et donc finalement $t.t1$ est tirée k unités de temps après $test$. Si $k \leq 2$ alors $t.t1$ est tirée avant $t.sup$ et donc $outnz$ est marquée k unités de temps après le tir de $test$. Si $k = 4$, alors $t.sup$ est tirée avant $t.t1$ et donc $outz$ est marquée k unités de temps après le tir de $test$. Par construction des autres motifs, nous ne pouvons pas avoir $2 < k < 4$. Par ailleurs, la phase ϕ_i n'est pas changée par ce motif. □

6.6.5 Instruction de décrémentation

Décrémenter le compteur c_i consiste à doubler la phase $\phi_{tick}^{i.Clock}$. Pour cela nous utilisons le motif de la figure 6.12 précédé par le motif de test précédent afin d'éviter que le compteur ne soit décrémenté s'il est nul : si c_i 0 alors $c_i = c_i - 1$.

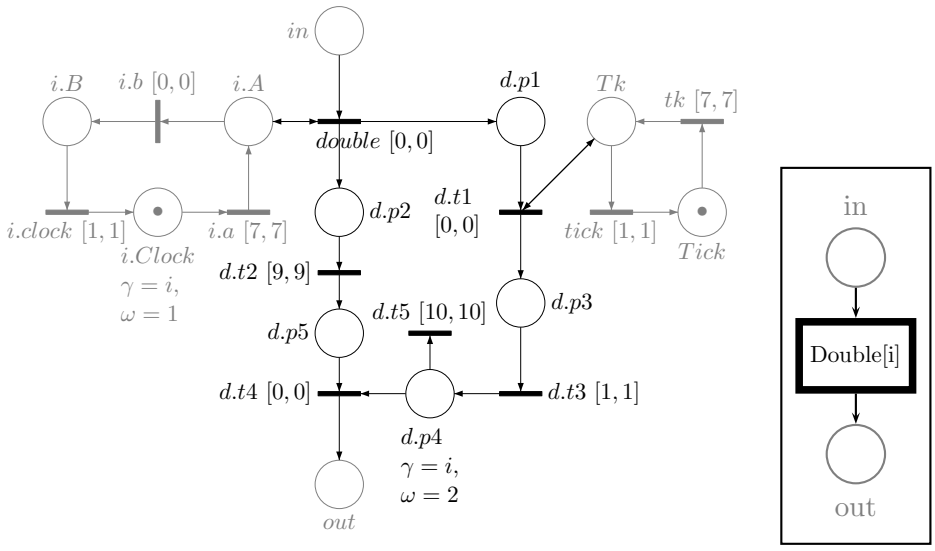

FIG. 6.12 – Motif de doublement de phase ($i \in \{1, 2\}$)

Lemme 3 (Doublement de phase)
$(B1)$ $in \wedge \phi_i(in) = k \wedge k \leq 2 \ \rightsquigarrow \ out \wedge \phi_i(out) = 2 * k$

Preuve. Supposons que in soit marquée et que $\phi_i(in) = k$ avec $0 < k \leq 2$ (Rappelons que k prend ses valeurs dans l'ensemble $\{4, 2, 1, \frac{1}{2} \ldots\}$). Comme pour le motif de test, $d.t1$ est tirée k unités de temps après $double$. Alors $d.t3$ et $tick$ sont tirées $k + 1$ unités de temps après $double$ et $i.clock$ une unité de temps après $double$. Supposons que $double$ soit tirée à la date d. $i.a$ est alors sensibilisée et active entre les dates $d+1$ et $d+k+1$ soit pendant k unités de temps. Puis elle devient inactive entre les dates $d + k + 1$ et $d + 9$. Finalement le prochain tir de $i.clock$ a lieu a la date $(d + 9) + 8 - k = d + 17 - k$ et le prochain tir de $tick$ suivant $i.clock$ a la date $(d + k + 1) + 16 = d + 17 + k$ soit $2 * k$ unités de temps après $i.clock$ (à la date $d + 9 + k$ la transition $i.a$ est toujours inactive). Par conséquent le déphasage $\phi_i(out)$ est $2 * k$. □

Lemme 4 (Décrementation du compteur)
$(B1)$ $in \wedge \phi_i(in) = k \wedge k \leq 2 \ \rightsquigarrow \ out \wedge \phi_i(out) = 2 * k$
$(B2)$ $in \wedge \phi_i(in) = 4 \ \rightsquigarrow \ out \wedge \phi_i(out) = 4$

Preuve. Cette preuve découle directement des lemmes 2 et 3. □

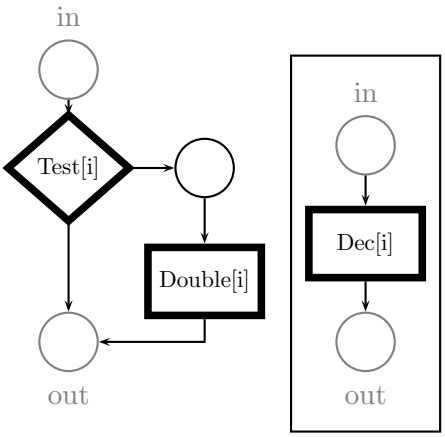

FIG. 6.13 – Instruction de décrémentation ($i \in \{1, 2\}$)

6.6.6 Instruction d'incrémentation

Pour incrémenter le compteur c_i, il faut diviser la phase correspondante par deux. Ceci est fait en choisissant une valeur au hasard pour le compteur temporaire c_3 grâce au motif de la figure 6.14. Cette valeur est ensuite multipliée par deux en utilisant le motif de la figure 6.12 et comparée à la valeur du compteur c_i en utilisant le motif de la figure 6.15. Si les deux valeurs sont égales, cela signifie que c_3 contient la valeur de c_i divisée par 2 et nous pouvons donc affecter la valeur de c_3 à c_i. Dans le cas contraire, nous recommençons le processus. Il existe une et une seule séquence de tirs telle que la bonne valeur soit « devinée ».

Lemme 5 (Sélection)
$(S1)$ $in \wedge \phi_i(in) = k \wedge 0 < k \leq 4 \ \rightsquigarrow$
$out \wedge \phi_i(out) = k \wedge \phi_3(out) = \phi_{guess}(out) = \delta$ *où* $0 \leq \delta \leq k$

Preuve. Supposons que *in* soit marquée et que $\phi_i(in) = k$ avec $0 < k \leq 4$. Supposons que *c.loop* est tirée à la date d. Alors *c.bad* sera tirée à la date $d + k$, et tout le processus recommencera, à moins que *c.choose* ne soit tirée avant, c'est-à-dire dans l'intervalle $[d, d+k]$. *tick* sera tirée à la date $d + k + 1$, et si *c.choose* est tirée, *c.init* sera tirée dans l'intervalle $[d + 1, d + k + 1]$. Par conséquent, la phase des événements 3.*clock* et *guess* est initialisée à une valeur comprise entre 0 et $k = \phi_i(in)$. Par ailleurs, ce motif ne change pas la phase de *i.clock*. □

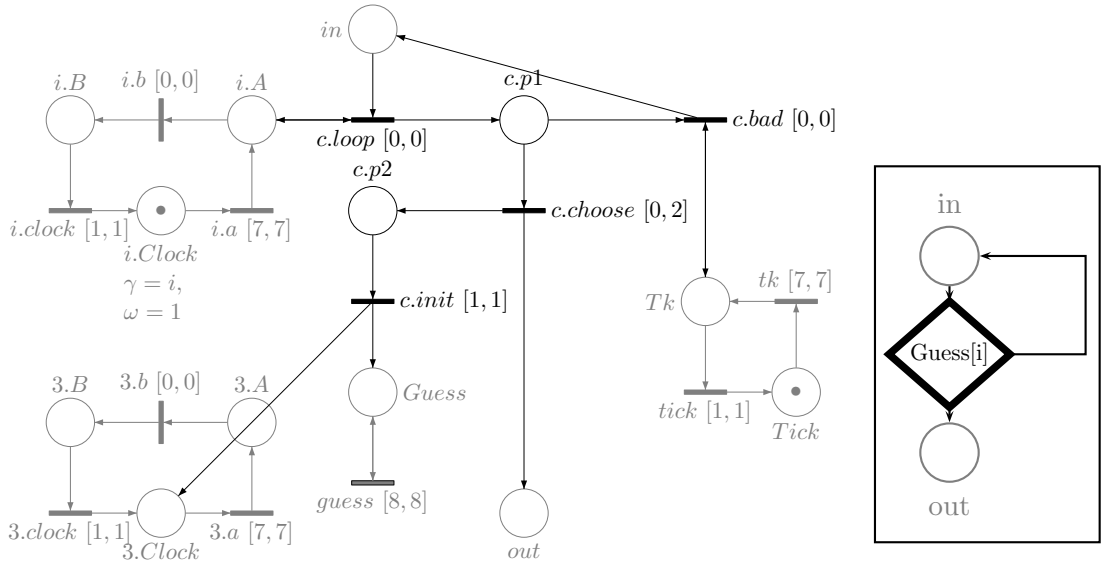

FIG. 6.14 – Motif de sélection ($i \in \{1,2\}$)

Lemme 6 (Comparaison et affectation)
$(C1)$ $in \wedge \phi_i(in) = \phi_3(in) \wedge 0 \leq \phi_3(in) \leq \phi_i(in) \leq 4 \ \rightsquigarrow$
$(out \wedge \phi_i(out) = \phi_{guess}(in)) \vee (retry \wedge \phi_i(retry) = \phi_i(in))$
$(C2)$ $in \wedge \phi_i(in) \neq \phi_3(in) \wedge 0 \leq \phi_3(in) \leq \phi_i(in) \leq 4 \ \rightsquigarrow \ retry \wedge \phi_i(retry) = \phi_i(in)$

Preuve. *a.equal* ne peut être tirée que si $\phi_i(in) = \phi_3(in)$, sinon, c'est *a.retry* qui est tirée et aucune phase n'est changée. Supposons donc que $\phi_i(in) = \phi_3(in)$. Supposons que *a.equal* est tirée à la date d, alors le prochain tir de *guess* aura lieu dans l'intervalle $[d, d+8]$ et donc *a.assign* sera toujours tirable en même temps que *guess*, c'est-à-dire que $\phi_i(out) = \phi_{guess}(out)$. □

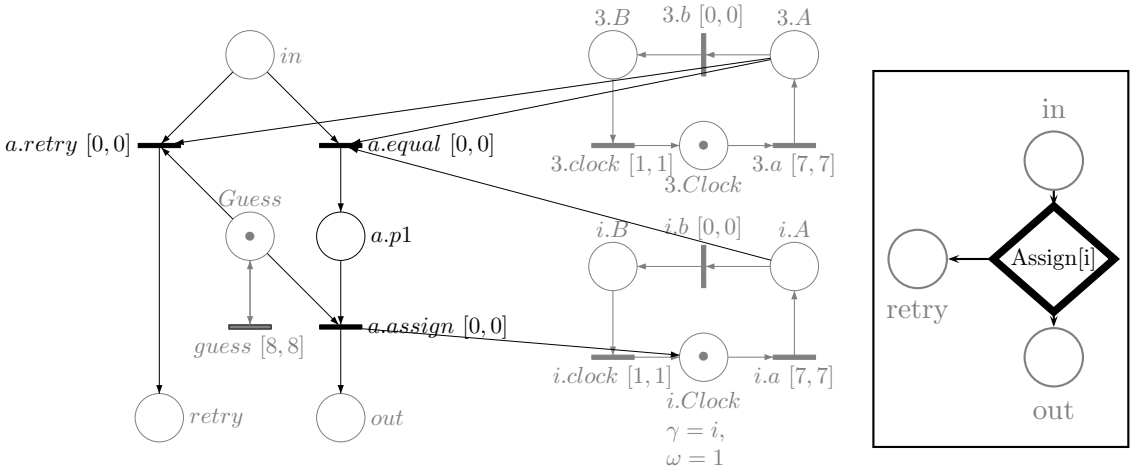

FIG. 6.15 – Motif de comparaison et d'affectation ($i \in \{1,2\}$)

Lemme 7 (Incrémentation de compteur)
$(I1)$ $in \wedge \phi_i(in) = k \wedge 0 < k \leq 4 \ \rightsquigarrow \ out \wedge \phi_i(out) = k/2$

Preuve. Cette preuve découle directement des lemmes 3, 5 et 6. □

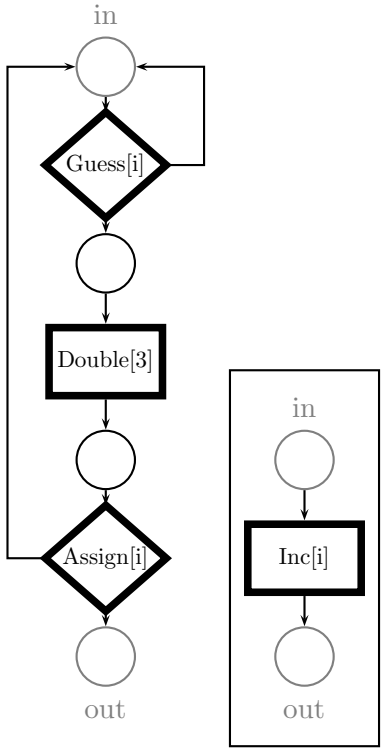

FIG. 6.16 – Instruction d'incrémentation ($i \in \{1, 2\}$)

6.6.7 Modélisation de la machine à deux compteurs

Soit une machine à deux compteurs $\mathcal{M}$. Nous construisons un réseau de Petri T-temporel étendu à l'ordonnancement $\mathcal{T}$ encodant $\mathcal{M}$ de la façon suivante :

- nous créons une place marquée *start* pour $\mathcal{T}$;
- pour chacun des $n + 1$ états $q_0, q_1, \ldots, q_n$ de $\mathcal{M}$, nous ajoutons une place non marquée $p_0, p_1, \ldots, p_n$;
- nous ajoutons les compteurs et le motif d'initialisation de la figure 6.10 de façon à ce que *start* soit la place *in* et p_0 la place *out* ;
- pour chaque instruction (q_a, dec_i, q_b) (respectivement (q_a, inc_i, q_b)) nous ajoutons un motif de décrémentation (respectivement d'incrémentation) connecté aux places p_a en entrée et p_b en sortie ;
- pour chaque instruction $(q_a, test_i, q_b, q_c)$, nous ajoutons à $\mathcal{T}$ un motif de test connecté aux places p_a en entrée, p_b en sortie *outz* et p_c en sortie *outnz*.

6.6.8 Indécidabilité

Soit $\mathcal{M} = (Q, q_0, q_F, \mathcal{I}, c_1, c_2)$ une machine à deux compteurs. Une *configuration* de $\mathcal{M}$ est un triplet (q, x_1, x_2) avec $q \in Q$ et $(x_1, x_2) \in \mathbb{N}^2$.

q représente l'état de la machine et x_1, x_2 la valeur des compteurs. La *configuration initiale* est $c_0 = (q_0, 0, 0)$.

Soit un état $s = (m, \nu)$ de la sémantique $\mathcal{S}_\mathcal{T}$ du réseau de Petri T-temporel étendu à l'ordonnancement $\mathcal{T}$ et une configuration $c = (q_i, x_1, x_2)$ de $\mathcal{M}$. Nous dirons que s encode c, et nous noterons $s \cong c$, si :

- m marque p_i, $1.Clock$, $2.Clock$ et $Tick$;
- $\nu(tick) - \nu(1.clock) = 1/2^{x_1+1}$ et $\nu(tick) - \nu(2.clock) = 1/2^{x_2+1}$.

Théorème 16 *Une configuration c est accessible depuis c_0 dans la machine $\mathcal{M}$, ce que nous écrivons $c_0 \xrightarrow{c}$, si et seulement si il existe un état s de la sémantique du réseau de Petri T-temporel étendu à l'ordonnancement $\mathcal{T}$ encodant $\mathcal{M}$ tel que s est accessible depuis l'état initial s_0 ($s_0 \xrightarrow{s}$) et $s \cong c$.*

Preuve. Nous allons procéder par induction. Soit s l'état de $\mathcal{T}$ obtenu après le tir de *init*. s est accessible depuis s_0 et il est clair que $s \cong c_0$.

Supposons maintenant que $\mathcal{M}$ est dans la configuration c accessible depuis c_0 et qu'il existe un état s de $\mathcal{T}$, accessible depuis s_0 tel que $s \cong c$.

- Soit $I \in \mathcal{I}$ une instruction de $\mathcal{M}$ et c' la configuration atteinte par $\mathcal{M}$ en exécutant I. Alors les lemmes 4, 7 et 2 impliquent qu'il existe un état s' obtenu en « exécutant » le motif correspondant à I tel que $s' \cong c'$.
- De la même façon, soit s' l'état de $\mathcal{T}$ obtenu en « exécutant » le motif correspondant à une instruction I. Si c' est une configuration telle que $s' \cong c'$ alors, les lemmes 4, 7 et 2 impliquent que c' est accessible depuis c par l'instruction I.

□

Théorème 17 *L'accessibilité d'état, l'accessibilité de marquage, la k-bornitude et la vivacité sont des problèmes indécidables pour les réseaux de Petri T-temporels étendus à l'ordonnancement même bornés.*

Preuve. Pour toute machine à deux compteurs $\mathcal{M}$, le réseau de Petri encodant $\mathcal{M}$ est borné (et même sauf).

- *Accessibilité d'état* : d'après le théorème 16, une configuration c contenant l'état q_F, donc tel que la machine s'arrête, est accessible ssi il existe un état $s \cong c$ accessible depuis s_0. Donc l'accessibilité d'état est indécidable.
- *Accessibilité de marquage* : supposons que $q_0 \neq q_F$. Soit $\mathcal{T}'$ le réseau obtenu en ajoutant une place p_{n+1} et une transition t au réseau telles que les places amont de t sont les places marquées dans $\mathcal{T}$ après le tir de *init*, sauf p_0 et la seule place aval est p_{n+1}. Alors un état avec le marquage $\{p_{n+1}\}$ est accessible ssi $\mathcal{M}$ s'arrête.
- *k-bornitude* : supposons que l'arc entre t et p_{n+1} soit de poids $k+1$, $k > 1$ au lieu de 1 alors $\mathcal{T}'$ est k-borné ssi $\mathcal{M}$ ne s'arrête pas.
- *Vivacité* : ce problème est équivalent à l'accessibilité de marquage. Il est donc également indécidable.

□

Les instructions de la machine ne sont jamais exécutées simultanément et nous savons affecter un déphasage à un événement, comme dans le motif « comparaison et affectation ». Nous pourrions donc facilement utiliser une seule copie des motifs de doublement et de sélection. Donc toute machine à deux compteurs peut être encodée dans un réseau de Petri T-temporel

étendu à l'ordonnancement ne comportant qu'un seul intervalle de tir non réduit à un point, deux processeurs et deux places affectées à chacun de ces processeurs.

Si tous les intervalles sont réduits à un point, l'espace d'états est fini [LR03a] et si nous retirons le processeur (ou une place affectée à ce processeur) nous sommes dans le cas des réseaux de Petri T-temporels bornés et donc les problèmes cités sont décidables.

6.7 Graphe des classes d'états

La première méthode de calcul de l'espace d'états des *Scheduling*-TPN a été proposée par Olivier H. Roux et Anne-Marie Déplanche sous la forme d'une adaptation de la méthode du graphe des classes d'états pour le calcul de l'espace d'états des *Scheduling*-TPN dans [RD02]. Cette méthode calcule une surapproximation de l'espace d'états. Nous allons présenter dans cette section une méthode de calcul exacte dont nous déduirons la surapproximation utilisant les DBM.

6.7.1 Classes d'états

La définition des classes d'états ne change pas par rapport à celle des réseaux de Petri T-temporels :

Définition 29 (Classe d'états) *Une* classe d'états *C, d'un réseau de Petri T-temporel étendu à l'ordonnancement, est un couple (M, D) où M est un marquage du réseau et D un polyèdre appelé* domaine de tir.

Le polyèdre D prend la forme générale suivante :

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

θ_i représente l'instant de tir de la transition sensibilisée t_i relativement *à l'instant où l'on est entré dans la classe.*

Nous avons montré dans [LR03a] que la forme générale des polyèdres présentée ici n'est pas réductible à une DBM. Ce résultat sera présenté dans la sous-section 6.7.3.

6.7.2 Successeurs d'une classe d'états

Dans un *Scheduling*-TPN seules les transitions actives sont potentiellement tirables. Cela se traduit par la définition suivante :

Définition 30 (Transition tirable depuis une classe) *Soit $C = (M, D)$ une classe d'états d'un réseau de Petri T-temporel étendu à l'ordonnancement. Une transition t_i est dite* tirable *depuis C ssi t_i est active et il existe une solution $(\theta_1^*, \ldots, \theta_n^*)$ de D, telle que $\forall j \in [\![1, n]\!] - \{i\}$, t.q. t_j est active , $\theta_i^* \leq \theta_j^*$.*

Soit une classe $C = (M, D)$ et une transition tirable t_f, la classe $C' = (M', D')$, successeur de C par t_f, est donnée par :

– Le nouveau marquage est calculé de façon classique par $M' = M - {}^\bullet t_f + t_f{}^\bullet$

– Le nouveau domaine de tir, D', est calculé selon les étapes suivantes. Nous le noterons $next(D, t_f)$

1. changements de variables $\forall j$, t.q. t_j est active, $\theta_j = \theta_f + \theta'_j$;
2. $\forall j \neq t_f$, ajout des contraintes $\theta'_j \geq 0$;
3. élimination des variables correspondant à des transitions désensibilisées par le tir de t_f (ce qui inclut donc t_f), par exemple en utilisant la méthode de Fourier-Motzkin [Dan63] ;
4. ajout des inéquations relatives aux transitions nouvellement sensibilisées par le tir de t_f :
$$\forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq \theta'_k \leq \beta(t_k).$$
5. détermination de la forme canonique D'^* du nouveau domaine de tir.

Nous voyons que pour les *Scheduling*-TPN, les changements de variables ne sont effectués que pour les variables correspondant à des transitions *actives*. Ce sont en effet les seules pour lesquelles le temps s'écoule. Le reste est inchangé. Notons qu'en toute rigueur les contraintes $\theta'_j \geq 0$ ne sont requises que pour les transitions t_j actives.

Cette différence avec les réseaux de Petri T-temporels classiques n'est cependant pas anodine, puisqu'elle entraîne la génération de domaines de tirs qui ne peuvent pas être représentés par des DBM.

6.7.3 Forme générale des domaines de tir

Considérons la figure 6.17. Le réseau qui y est représenté constitue un contre-exemple qui prouve le théorème suivant :

Théorème 18 *Les domaines des classes d'états d'un réseau de Petri T-temporel étendu à l'ordonnancement ne peuvent pas être représentés de façon exacte par des DBM.*

Étant donnés les résultats d'indécidabilité que nous avons prouvés, ce théorème n'est pas étonnant.

Après le tir de la séquence t_4, t_1, t_5, la transition t_6 n'est pas tirable car soit t_2 soit t_3 *doit* être tirée avant elle, en fonction de l'instant de tir de t_4. La classe obtenue est :

$$\begin{cases} \{p_2, p_3, p_6\}, \\ \begin{cases} 0 \leq \theta_2 \leq 4, \\ 0 \leq \theta_3 \leq 4, \\ 4 \leq \theta_6 \leq 4, \\ 1 \leq \theta_2 + \theta_3 \leq 6 \end{cases} \end{cases}$$

Nous pouvons constater que dans cette classe, t_6 n'est effectivement pas tirable car, si elle l'était, nous aurions $\theta_2 = \theta_3 = \theta_6 = 4$ et donc $\theta_2 + \theta_3 = 8$. Cela prouve également que la dernière ligne du domaine n'est pas redondante avec le reste du domaine car si nous supprimons la contrainte $\theta_2 + \theta_3 \leq 6$, t_6 devient tirable.

Ces contraintes, $1 \leq \theta_2 + \theta_3 \leq 6$ ne peuvent pas être représentées par une DBM. Le polyèdre représentant le domaine la classe étant donné sous forme minimale (unique), cela prouve qu'il n'existe pas de DBM pouvant le représenter exactement et donc cela prouve le théorème.

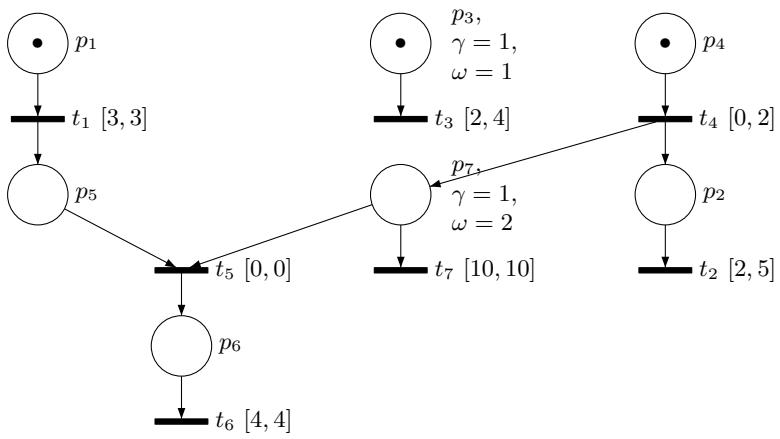

FIG. 6.17 – Un réseau de Petri T-temporel étendu à l'ordonnancement

De la même façon, nous pourrions trouver des réseaux conduisant à des contraintes plus compliquées, impliquant plus de deux variables avec toutes les combinaisons de signes. Illustrons cela sur un exemple formel :

Soit $C = (M, D)$ une classe d'états d'un *Scheduling*-TPN, telle que :

$$D = \begin{cases} \alpha_1 \leq \theta_1 \leq \beta_1, \\ \alpha_2 \leq \theta_2 \leq \beta_2, \\ \alpha_3 \leq \theta_3 \leq \beta_3, \\ \alpha_4 \leq \theta_4 \leq \beta_4, \\ -\gamma_{21} \leq \theta_1 - \theta_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq \theta_1 - \theta_3 \leq \gamma_{13}, \\ -\gamma_{41} \leq \theta_1 - \theta_4 \leq \gamma_{14}, \\ -\gamma_{32} \leq \theta_2 - \theta_3 \leq \gamma_{23}, \\ -\gamma_{42} \leq \theta_2 - \theta_4 \leq \gamma_{24}, \\ -\gamma_{43} \leq \theta_3 - \theta_4 \leq \gamma_{34} \end{cases} \tag{6.1}$$

Supposons que t_1, t_2 et t_3 (correspondant aux variables θ_1, θ_2, θ_3) sont actives, et que t_4 ne l'est pas. Supposons enfin que t_1 est tirable.

Calculons le domaine de la classe obtenue par le tir de t_1. La première étape consiste à effectuer les changements de variables $\theta_i \leftarrow \theta'_i + \theta_1$ pour toutes les transitions actives sauf t_1.

Le domaine devient :

$$\begin{cases} \alpha_1 \leq \theta_1 \leq \beta_1, \\ \alpha_2 \leq \theta_1 + \theta'_2 \leq \beta_2, \\ \alpha_3 \leq \theta_1 + \theta'_3 \leq \beta_3, \\ \alpha_4 \leq \theta'_4 \leq \beta_4, \\ -\gamma_{21} \leq \theta_1 - \theta_1 - \theta'_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq \theta_1 - \theta_1 - \theta'_3 \leq \gamma_{13}, \\ -\gamma_{41} \leq \theta_1 - \theta_4 \leq \gamma_{14}, \\ -\gamma_{32} \leq \theta_1 + \theta'_2 - \theta_1 - \theta'_3 \leq \gamma_{23}, \\ -\gamma_{42} \leq \theta_1 + \theta'_2 - \theta_4 \leq \gamma_{24}, \\ -\gamma_{43} \leq \theta_1 + \theta'_3 - \theta_4 \leq \gamma_{34} \end{cases} \tag{6.2}$$

Ensuite nous rajoutons les contraintes $\forall j, \theta'_j \geq 0$. Nous en profitons pour réécrire le système sous la forme de majorants et de minorants de θ_1 afin de préparer l'élimination de cette variable par la méthode de Fourier-Motzkin :

$$\begin{cases} \alpha_1 \leq \theta_1, & \theta_1 \leq \beta_1, \\ \alpha_2 - \theta'_2 \leq \theta_1, & \theta_1 \leq \beta_2 - \theta'_2, \\ \alpha_3 - \theta'_3 \leq \theta_1, & \theta_1 \leq \beta_3 - \theta'_3, \\ -\gamma_{41} + \theta_4, \leq \theta_1 & \theta_1 \leq \gamma_{14} + \theta_4, \\ -\gamma_{42} + \theta_4 - \theta'_2 \leq \theta_1, & \theta_1 \leq \gamma_{24} + \theta_4 - \theta'_2, \\ -\gamma_{43} + \theta_4 - \theta'_3 \leq \theta_1, & \theta_1 \leq \gamma_{34} + \theta_4 - \theta'_3, \\ \alpha_4 \leq \theta'_4 \leq \beta_4, & \theta'_2 \geq 0, \\ -\gamma_{32} \leq \theta'_2 - \theta'_3 \leq \gamma_{23}, & \theta'_3 \geq 0, \\ -\gamma_{21} \leq -\theta'_2 \leq \gamma_{12}, & \\ -\gamma_{31} \leq -\theta'_3 \leq \gamma_{13} & \end{cases} \tag{6.3}$$

La méthode de Fourier-Motzkin consiste alors à écrire que le système a des solutions si et seulement si les minorants de θ_1 sont inférieurs ou égaux à ses majorants. Le nouveau système est équivalent au précédent. Après quelques simplifications, nous obtenons :

$$\begin{cases} \max\{0, -\gamma_{12}\} \leq \theta'_2 \leq \gamma_{21}, \\ \max\{0, -\gamma_{13}\} \leq \theta'_3 \leq \gamma_{31}, \\ \alpha_4 \leq \theta_4 \leq \beta_4, \\ -\gamma_{32} \leq \theta'_2 - \theta'_3 \leq \gamma_{23}, \\ -\gamma_{42} - \beta_1 \leq \theta'_2 - \theta_4 \leq \gamma_{24} - \alpha_1, \\ -\gamma_{43} - \beta_1 \leq \theta'_3 - \theta_4 \leq \gamma_{34} - \alpha_1 \\ \alpha_2 - \gamma_{14} \leq \theta'_2 + \theta_4 \leq \beta_2 + \gamma_{41}, \\ \alpha_3 - \gamma_{14} \leq \theta'_3 + \theta_4 \leq \beta_3 + \gamma_{41}, \\ \alpha_2 - \gamma_{34} \leq \theta'_2 + \theta_4 - \theta'_3 \leq \beta_2 + \gamma_{43}, \\ \alpha_3 - \gamma_{24} \leq \theta'_3 + \theta_4 - \theta'_2 \leq \beta_3 + \gamma_{42} \end{cases} \tag{6.4}$$

Nous pouvons voir que les quatre dernières lignes contiennent huit inéquations qui ne peuvent pas être représentées par une DBM. Il est possible que ces inéquations soient redondantes mais dans le cas général ce n'est pas le cas, comme nous l'avons montré avec le réseau de la figure 6.17. Par ailleurs, nous pouvons facilement voir que le tir d'une autre transition à partir d'un domaine de cette forme pourrait conduire à des contraintes encore plus complexes, c'est-à-dire impliquant un nombre plus élevé de variables. Nous obtenons donc un polyèdre de forme générale.

6.7.4 Graphe des classes d'états

Pour un *Scheduling*-TPN $\mathcal{T}$, outre la forme particulière des classes d'états, la définition du graphe des classes d'états ne change pas. Comme au chapitre 4, avec la nouvelle règle de tirabilité des transitions et le nouveau calcul des successeurs, nous définissons le système de transitions $\Gamma'(\mathcal{T})$:

$\Gamma'(\mathcal{T}) = (\mathbf{C}, C_0, \rightarrow)$ où

- $\mathbf{C} = \mathbb{N}^P \times \mathbb{R}^T$;
- $C_0 = (M_0, D_0)$, où M_0 est le marquage initial du réseau et $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$;
- $\rightarrow \in \mathbf{C} \times T \times \mathbf{C}$ est la relation de transition définie par :

$$(M, D) \xrightarrow{t} (M', D') \text{ ssi } \begin{cases} M' = M - {}^\bullet t + t^\bullet, \\ t \text{ est tirable depuis } (M, D), \\ D' = next(D, t). \end{cases}$$

Nous définissons également l'égalité de deux classes d'états :

Définition 31 (Egalité de classes d'états) *Deux classes d'états $C = (M, D)$ et $C' = (M', D')$ d'un réseau de Petri T-temporel étendu à l'ordonnancement sont dites* égales *si $M = M'$ et $[\![D]\!] = [\![D']\!]$. Nous notons $C \equiv C'$.*

On définit alors le graphe des classes $\Gamma(\mathcal{T})$ du réseau de Petri T-temporel étendu à l'ordonnancement $\mathcal{T}$ par le quotient de $\Gamma'(\mathcal{T})$ par la relation d'équivalence $\equiv$: $\Gamma(\mathcal{T}) = \Gamma'(\mathcal{T})/\equiv$.

Cependant, une conséquence du théorème 17 est que ce quotient n'est pas forcément fini.

6.7.5 Surapproximation

Au vu de la forme générale des domaines de tir des classes d'états des *Scheduling*-TPN, nous voyons que le calcul de ces classes implique la manipulation de polyèdres généraux ce qui est bien plus coûteux que la manipulation de DBM, la complexité étant exponentielle en le nombre de variables contre polynomiale pour les DBM.

Par conséquent, nous proposons une surapproximation naturelle des classes d'états qui consiste à supprimer du domaine toutes les inéquations ne pouvant pas être représentées par des DBM. Le domaine représenté par le système d'inéquations (6.4) devient alors :

$$\begin{cases} \max\{0, -\gamma_{12}\} \leq \theta'_2 \leq \gamma_{21}, \\ \max\{0, -\gamma_{13}\} \leq \theta'_3 \leq \gamma_{31}, \\ \alpha_4 \leq \theta_4 \leq \beta_4, \\ -\gamma_{32} \leq \theta'_2 - \theta'_3 \leq \gamma_{23}, \\ -\gamma_{42} - \beta_1 \leq \theta'_2 - \theta_4 \leq \gamma_{24} - \alpha_1, \\ -\gamma_{43} - \beta_1 \leq \theta'_3 - \theta_4 \leq \gamma_{34} - \alpha_1 \end{cases} \tag{6.5}$$

Nous avons vu pour le réseau de la figure 6.17 que cela peut conduire à générer des classes qui ne correspondent pas à des états du réseau en rendant tirables des transitions qui ne le sont pas.

Nous pouvons caractériser les cas où la surapproximation relâche effectivement des contraintes par rapport au calcul exact. Dans [BFSV04], Bucci *et al.* proposent la condition nécessaire et suffisante suivante : soit $C' = (M', D')$ la classe obtenue par le tir de t_f depuis la classe $C = (M, D)$. Le remplacement du domaine exact par sa surapproximation relâche des contraintes si et seulement si la classe parente C contient à la fois des transitions actives et des transitions sensibilisées mais non actives qui restent sensibilisées après le tir de t_f.

En fait, nous n'avons là qu'une condition nécessaire : si la surapproximation relâche des contraintes alors la classe parente contient des transitions actives et des transitions non actives qui restent sensibilisées par le tir de t_f.

Supposons que D, sous forme canonique, est donné par le système d'inéquations (6.1), t_1, t_2 et t_3 étant actives et t_4 ne l'étant pas. Il est assez facile de se rendre compte que cette configuration est représentative de tous les domaines de tirs représentables par des DBM. En effet, en ajoutant des variables actives dans D nous dupliquons, dans D' (obtenu par le tir de t_1), les inéquations relatives à θ_3 (ou θ_2) et en ajoutant des variables non actives, nous dupliquons les inéquations relatives à θ_4.

Comme précédemment, le tir de t_1 conduit au système (6.4) :

$$\begin{cases} \max\{0, -\gamma_{12}\} \leq \theta'_2 \leq \gamma_{21}, \\ \max\{0, -\gamma_{13}\} \leq \theta'_3 \leq \gamma_{31}, \\ \alpha_4 \leq \theta_4 \leq \beta_4, \\ -\gamma_{32} \leq \theta'_2 - \theta'_3 \leq \gamma_{23}, \\ -\gamma_{42} - \beta_1 \leq \theta'_2 - \theta_4 \leq \gamma_{24} - \alpha_1, \\ -\gamma_{43} - \beta_1 \leq \theta'_3 - \theta_4 \leq \gamma_{34} - \alpha_1 \\ \alpha_2 - \gamma_{14} \leq \theta'_2 + \theta_4 \leq \beta_2 + \gamma_{41}, \\ \alpha_3 - \gamma_{14} \leq \theta'_3 + \theta_4 \leq \beta_3 + \gamma_{41}, \\ \alpha_2 - \gamma_{34} \leq \theta'_2 + \theta_4 - \theta'_3 \leq \beta_2 + \gamma_{43}, \\ \alpha_3 - \gamma_{24} \leq \theta'_3 + \theta_4 - \theta'_2 \leq \beta_3 + \gamma_{42} \end{cases}$$

Nous pouvons remarquer que les contraintes « non DBM » présentes sur les quatre dernières lignes font toutes intervenir au moins une transition qui était active dans C (t_2 ou t_3) et une transition qui était non active (t_4). Par conséquent, la preuve de la condition nécessaire précédente est immédiate : si après le tir de t_1, il ne reste pas au moins une transition qui était active et une transition qui était inactive, il n'y a pas de contraintes « non DBM ».

Par ailleurs la réciproque est fausse : si, par exemple, $\gamma_{24} = \beta_2 - \alpha_4$ et $\gamma_{41} = \beta_4 - \alpha_1$ (c'est-à-dire que ces contraintes étaient redondantes dans D, ce qui est le cas, par exemple, pour la classe initiale) alors l'inéquation $\theta'_2 + \theta_4 \leq \beta_2 + \gamma_{41}$ peut être obtenue comme la somme de $\theta'_2 \leq \gamma_{21}$ et $\theta_4 \leq \beta_4$. Elle est donc redondante et le même raisonnement est applicable aux autres contraintes[2].

Finalement, nous pouvons donner une condition nécessaire et suffisante :

Théorème 19 (Surapproximation effective) *Soit $C = (M, D)$ une classe d'états d'un Scheduling-TPN telle que, sous forme canonique, $D = \{\alpha_i \leq \theta_i \leq \beta_i, \theta_i - \theta_j \leq \gamma_{ij}\}$. Soit t_f une transition tirable depuis C et dont le tir conduit à la classe $C' = (M', D')$. Soit $\overline{D'}$ le domaine surapproximé sous forme de DBM obtenu à partir de D'.*
$\overline{D'}$ relâche des contraintes de D ssi $\exists i \in enabled(Act(M)), \exists j \in enabled(M) - enabled(Act(M))$ t.q. $i, j \notin disabled(M, t_f)$ et $\beta_j + \gamma_{if} > \beta_i + \gamma_{jf} \vee \alpha_j - \gamma_{fi} < \alpha_i - \gamma_{fj}$ ou $\exists i, k \in enabled(Act(M)), \exists j \in enabled(M) - enabled(Act(M))$ t.q. $i, j, k \notin disabled(M, t_f)$ et $\beta_j + \gamma_{ik} > \beta_i + \gamma_{jk} \vee \alpha_j - \gamma_{ki} < \alpha_i - \gamma_{kj}$.

En ajoutant des transitions tirables, cette approximation présente le risque de rendre non borné le marquage des classes d'états et donc leur nombre. Cependant, excepté le risque d'être pessimiste, la surapproximation n'est pas gênante pour la vérification de propriétés

[2]Cela implique notamment qu'il est impossible d'obtenir des contraintes « non DBM » dans le domaine d'une classe issue directement de la classe initiale.

de sûreté. Par ailleurs, nous avons vu que le problème de l'accessibilité pour les *Scheduling*-TPN est indécidable. Par conséquent, rendre le graphe des classes d'états non fini par la surapproximation n'est pas un risque rédhibitoire.

6.8 Conclusion

Dans ce chapitre, nous avons présenté le modèle des réseaux de Petri T-temporels étendus à l'ordonnancement. Nous avons illustré son expressivité et démontré l'indécidabilité de la plupart des problèmes intéressants concernant ce modèle. Nous avons donc proposé un semi-algorithme de calcul de l'espace d'états adapté du calcul du graphe des classes d'états classique. Ce calcul étant très coûteux en temps de calcul nous avons également proposé une surapproximation des classes d'états de ce modèle, à base de DBM.

Nous allons voir dans le chapitre suivant comment utiliser ces résultats afin d'obtenir une méthode de vérification des réseaux de Petri T-temporels étendus à l'ordonnancement à l'aide de l'outil HyTech.

Chapitre 7

Automate à chronomètres des classes d'états

Dans ce chapitre, nous allons adapter la méthode développée au chapitre 5 permettant de traduire un réseau de Petri T-temporel en automate temporisé, au modèle étendu à l'ordonnancement [1].

Cependant, comme nous l'avons vu au chapitre précédent, les réseaux de Petri T-temporels étendus à l'ordonnancement n'appartiennent pas à la même classe de modèles que les réseaux de Petri T-temporels classiques ; ils disposent en plus de « l'effet *stopwatch* ». Un certain nombre de travaux, incluant la méthode du chapitre 5, ont montré comment traduire un réseau de Petri T-temporel borné en automate temporisé, en calculant l'espace d'états [GRR03, LR03b] ou de façon structurelle [CR03, CR04]. Des travaux sont en cours pour obtenir une traduction structurelle inverse, c'est-à-dire des automates temporisés vers les réseaux de Petri temporels bornés. Ce résultat achèverait de démontrer que les réseaux de Petri T-temporels et les automates temporisés appartiennent à la même classe de modèles[2].

Même sans ce résultat, l'accessibilité d'états étant indécidable pour les Scheduling-TPN, il est impossible de trouver, pour chaque Scheduling-TPN, un automate temporisé qui lui est temporellement bisimilaire, car cela impliquerait que l'accessibilité d'états est indécidable pour les automates temporisés. Par ailleurs, il est facile de se rendre compte qu'il n'existe pas d'automate temporisé acceptant le langage $\forall n \in [0,1], \forall m \geq 0, (t_1, n)(t_2, m)(t_3, 1-n)$ du réseau de Petri T-temporel étendu à l'ordonnancement de la figure 6.9 du chapitre 6.

Par conséquent, il est inutile d'espérer obtenir un automate temporisé par l'adaptation de la méthode du chapitre 5 aux réseaux de Petri T-temporels étendus à l'ordonnancement. Pour prendre en compte « l'effet *stopwatch* », nous allons donc traduire les réseaux de Petri T-temporels étendus à l'ordonnancement en automates à chronomètres.

Ces automates à chronomètres pourront ensuite être vérifiés à l'aide de l'outil HyTech[HHWT97]. Comme au chapitre 5, le lecteur pourra légitimement se demander pourquoi ne pas modéliser le système directement sous la forme d'automates à chronomètres. Les argument avancés précédemment sont toujours valables, à savoir que certaines caractéristiques des systèmes temps réel s'expriment de façon plus naturelle et - ou - de façon plus concise sous la forme d'un réseau de Petri T-temporel étendu à l'ordonnancement que sous la forme d'un automate à chronomètres. En particulier, l'ordonnanceur doit être modélisé explicitement pour les au-

[1] Ce travail a fait l'objet d'une publication dans [LR04].

[2] Toujours au sens de l'expressivité en termes d'acceptation de langages temporisés.

tomates à chronomètres alors qu'il est implicite avec les réseaux de Petri T-temporels étendus à l'ordonnancement car son comportement est inclus dans la sémantique du modèle. Par ailleurs, la méthode permet une modélisation utilisant conjointement les deux modèles. Cependant, nous disposons avec cette méthode d'un argument plus fort. En effet, une modélisation générique d'un système temps réel directement sous la forme d'automates à chronomètres requiert typiquement de faire le produit d'un automate par tâche ainsi qu'un automate par ordonnanceur (donc par processeur) ce qui nous donne au moins autant de chronomètres. Or, la complexité de la vérification d'un automate hybride est exponentielle en le nombre de chronomètres et par conséquent quand ce nombre augmente, le calcul de l'espace d'états est rapidement infaisable avec HyTech.

Le précalcul qui transforme un réseau de Petri T-temporel étendu à l'ordonnancement en automate à chronomètres est comme au chapitre 5 conçu de façon à réduire le nombre de chronomètres de l'automate résultat au maximum. Cette réduction est d'autant plus nécessaire qu'il n'existe pas de méthode syntaxique de réduction d'horloge du type de [DY96] pour les automates à chronomètres.

Enfin, et surtout, nous pouvons faire ce précalcul en utilisant la surapproximation du chapitre précédent et obtenir quand même un automate à chronomètres bisimilaire temporellement au réseau de Petri T-temporel étendu à l'ordonnancement initial car les localités ajoutées par la surapproximation ne sont en fait pas accessibles dans l'automate résultat.

En résumé, cette méthode propose, à partir d'un réseau de Petri T-temporel étendu à l'ordonnancement, de calculer rapidement (car le calcul est basé sur des DBM) un[3] automate à chronomètres temporellement bisimilaire possédant moins de chronomètres qu'une modélisation directe, permettant ainsi d'augmenter de façon importante la taille des systèmes vérifiables en pratique par HyTech.

7.1 Automates à chronomètres

Les automates à chronomètres (*stopwatch automata* ou SWA, en anglais) peuvent être vus comme une extension des automates temporisés à l'aide de chronomètres, c'est-à-dire d'horloges pour lesquelles l'écoulement du temps peut être suspendu puis repris plus tard.

7.1.1 Exemple

La figure 7.1 étend l'automate temporisé de la figure 5.1 en transformant l'horloge x en un chronomètre. Dans la localité l_0 le chronomètre x est actif ; le temps s'écoule pour elle. Dans la localité l_1 le chronomètre x est arrêté ; le temps ne progresse plus donc l'invariant restera toujours vérifié.

7.1.2 Définitions

Définition 32 (Automate à chronomètres) *Un* automate à chronomètres *est un 7-uplet* $(L, l_0, X, A, E, Inv, Dif)$ *où :*

- L *est un ensemble fini de* localités *;*
- l_0 *est la* localité initiale *;*
- X *est un ensemble fini de*chronomètres *à valeurs réelles positives ;*

[3] Le résultat est un unique automate et non pas un produit d'automates.

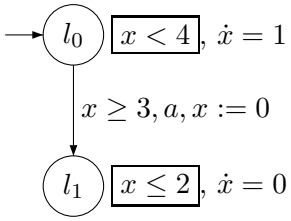

FIG. 7.1 – Un automate à chronomètres

- *A est un ensemble fini d'*actions *;*
- *$E \subset L \times \mathcal{C}(X) \times A \times 2^X \times 2^{X^2} \times L$ est un ensemble fini d'*arêtes*. Soit $e = (l, \delta, \alpha, R, \rho, l') \in E$. e est l'arête reliant la localité l à la localité l', avec la* garde *δ, l'action α, l'ensemble des chronomètres à remettre à zéro R et la fonction d'affectation de chronomètres ρ.*
- *$Inv \in \mathcal{C}(X)^L$ associe un* invariant *à chaque localité ;*
- *$Dif \in (\{0,1\}^X)^L$ associe à chaque localité l'*activité *des chronomètres dans cette localité. $\dot{X}$ désigne l'ensemble des dérivées par rapport au temps des variables de X : $\dot{X} = (Dif(l)x)_{x \in X}$.*

Dans un souci de simplicité, étant donnés une localité l, un chronomètre x et $b \in \{0,1\}$, nous noterons $Dif(l)(x) = b$ par $\dot{x} = b$ quand la localité considérée n'est pas ambiguë.

Définition 33 (Sémantique d'un SWA) *La sémantique d'un automate à chronomètres $\mathcal{A}$ est définie sous la forme d'un système de transitions temporisé $\mathcal{S}_\mathcal{A} = (Q, Q_0, \rightarrow)$ où*

- *$Q = L \times (\mathbb{R}^+)^X$;*
- *$Q_0 = (l_0, \overline{0})$;*
- *$\rightarrow \in Q \times (A \cup \mathbb{R}) \times Q$ est la relation définie pour $a \in A$ et $d \in \mathbb{R}^+$, par :*
  - *la relation de transition* discrète *: $(l, \nu) \xrightarrow{a} (l', \nu')$ ssi $\exists (l, \delta, a, R, \rho, l') \in E$ tel que*
    $$\begin{cases} \delta(\nu) = \texttt{true}, \\ \nu' = \nu[R \leftarrow 0][\rho], \\ Inv(l')(\nu') = \texttt{true} \end{cases}$$
  - *la relation de transition* continue *: $(l, \nu) \xrightarrow{d} (l, \nu')$ ssi* $\begin{cases} \nu' = \nu + \dot{X} * d, \\ \forall d' \in [0, d], Inv(l)(\nu + \dot{X} * d') = \texttt{true} \end{cases}$

7.2 Automate à chronomètres des classes d'états

Nous allons maintenant montrer comment traduire un réseau de Petri T-temporel borné en un automate à chronomètres, par le calcul de son espace d'états[4].

7.2.1 Classes d'états étendues

De la même façon qu'au chapitre 5, nous étendons les classes d'états des réseaux de Petri T-temporels étendus à l'ordonnancement avec des informations temporelles supplémentaires. Bien sûr il s'agit maintenant non plus d'horloges mais de chronomètres.

[4]Si ce calcul est possible, ce qui est indécidable.

Définition 34 (Classe d'états étendues) *Une classe d'états étendue, d'un réseau de Petri T-temporel étendu à l'ordonnancement, est un 4-uplet $(M, D, \chi, trans)$, où M est un marquage, D un polyèdre appelé* domaine de tir*, χ un ensemble de chronomètres et $trans \in (2^T)^\chi$ associe à chaque chronomètre un ensemble de transitions.*

7.2.2 Successeur d'une classes d'états étendue

Puisque nous avons désormais des chronomètres et non plus des horloges, il faut effectuer un certain nombre d'étapes supplémentaires dans l'algorithme de calcul des successeurs d'une classe d'états étendue. Ces étapes visent à calculer l'activité de tous les chronomètres car celle-ci est susceptible de changer en fonction de l'activité des transitions. Elles s'assurent également de la cohérence de l'association entre les transitions et les chronomètres qui les représentent ; un chronomètre actif (respectivement inactif) peut représenter uniquement les valuations de transitions actives (respectivement inactives).

Après les étapes définies au chapitre 5, il faut donc effectuer les étapes suivantes :

1. si toutes les transitions associées à un chronomètre actif (respectivement inactif) x sont inactives (respectivement actives) alors celui-ci est stoppé : $\dot{x} = 0$ (respectivement activé : $\dot{x} = 1$) ;
2. si l'image par $trans$ d'un chronomètre actif (respectivement inactif) x contient à la fois des transitions actives et des transitions inactives, alors un nouveau chronomètre x' est créé, de la même façon que pour les transitions nouvellement sensibilisées, auquel sont associées par $trans$ les transitions inactives (respectivement actives). Ce chronomètre est inactif (respectivement actif) et il prend la valeur de x : $x' = x$.

7.2.3 Graphe des classes d'états étendues

La classe initiale est définie par le marquage initial, le domaine $\{\alpha(t_i) \leq \theta_i \leq \beta(t_i)\}$, et les chronomètres x_0 et x_1 associées par $trans$ respectivement aux transitions actives et non actives. Bien sûr, s'il n'y a pas de transition non active, la classe initiale ne comporte qu'un seul chronomètre : x_0.

Nous pouvons définir, avec la classe initiale et la relation de transition définie ci-dessus, le graphe des classes d'états étendues $\Delta'(\mathcal{T})$ du réseau de Petri T-temporel étendu à l'ordonnancement $\mathcal{T}$. Nous définissons d'abord le système de transitions $\Delta''(\mathcal{T}) = (C^{ext}, C_0, \rightarrow^{ext})$:

- $C^{ext} = \mathbb{N}^P \times \mathbb{R}^T \times 2^X \times (2^T)^X$, X étant l'ensemble de tous les chronomètres ;
- $C_0 = (M_0, D_0, \chi_0, trans_0)$, où M_0 est le marquage initial, $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$, $\chi_0 = \{x_0, x_1\}$ et $trans_0 = ((x_0, enabled(Act(M_0))), (x_1, enabled(M_0) - enabled(Act(M_0))))$;
- $\rightarrow^{ext} \in C^{ext} \times T \times C^{ext}$ est la relation de transitions définies par les règles ci-dessus.

Comme précédemment ce système de transitions est potentiellement infini. Nous allons donc en faire le quotient avec une relation d'équivalence adéquate, *stopwatch-similarité* :

Définition 35 (Stopwatch-similarité) *Deux classes d'états étendues $C = (M, D, \chi, trans)$ et $C' = (M', D', \chi', trans')$ sont* stopwatch-similaires*, ce que nous notons $C \approx C'$, si elles ont le même marquage, le même nombre de chronomètres et si leurs chronomètres sont associés aux mêmes transitions :*

$$C \approx C' \Leftrightarrow \begin{cases} M = M', \\ |\chi| = |\chi|, \\ \forall x \in \chi, \exists x' \in \chi', trans(x) = trans'(x'). \end{cases}$$

Nous pouvons constater que cette relation d'équivalence est l'adaptation directe de la clock-similarité du chapitre 5.

D'un point de vue théorique, nous pouvons définir, de la même façon que précédemment, le graphe des classes d'états étendues $\Delta'(\mathcal{T})$ d'un réseau de Petri T-temporel étendu à l'ordonnancement $\mathcal{T}$ comme le quotient de $\Delta''(\mathcal{T})$ par la relation de stopwatch-similarité : $\Delta'(\mathcal{T}) = \Delta''(\mathcal{T})/\approx$.

D'un point de vue pratique, nous pouvons, de la même façon, arrêter le calcul des successeurs lors d'une inclusion de classes d'états étendues définie par :

Définition 36 (Inclusion de classes d'états étendues) *Une classe d'états étendue* $C' = (M', D', \chi', trans')$ *est* incluse *dans une classe d'états étendue* $C = (M, D, \chi, trans)$ *si* C *et* C' *sont stopwatch-similaires et* $[\![D']\!] \subset [\![D]\!]$. *Nous notons* $C' \subset C$.

Bien sûr, l'indécidabilité du problème de l'accessibilité pour les réseaux de Petri T-temporels étendus à l'ordonnancement implique que le calcul de ce graphe des classes étendues ne se termine pas forcément.

7.2.4 Automate à chronomètres des classes d'états

À partir du graphe des classes étendues nous définissons syntaxiquement l'automate à chronomètres des classes d'états :

Définition 37 (Automate à chronomètres des classes d'états) *L'*automate à chronomètres des classes d'états $\Delta(\mathcal{T}) = (L, l_0, X, A, E, Inv)$ *est défini syntaxiquement à partir du graphe des classes d'états étendues par :*

- L, *l'ensemble des localités, est l'ensemble des classes d'états étendues* C^{ext} ;
- l_0 *est la classes d'états étendue initiale* $(M_0, D_0, \chi_0, trans_0)$;
- $X = \bigcup_{(M,D,\chi,trans)\in C^{ext}} \chi$
- $A = T$ *est l'ensemble des transitions du réseau de Petri T-temporel*
- E *est l'ensemble des arêtes défini comme suit :*

$$\forall C_i = (M_i, D_i, \chi_i, trans_i), C_j = (M_j, D_j, \chi_j, trans_j) \in C^{ext},$$
$$\exists C_i \overset{t}{\rightarrow}^{ext} C_j \Leftrightarrow \exists (l_i, \delta, a, R, \rho, l_j) \ t.q. \begin{cases} \delta = ({trans_i}^{-1}(t) \geq \alpha(t)), \\ a = t, \\ R = {trans_j}^{-1}(\uparrow enabled(M_i, t)), \\ \forall x \in \chi_i, x' \in \chi_j, \\ \ t.q.\ trans_i(x) = trans_j(x') \\ \ et\ x' \notin R, \rho(x) = x' \end{cases}$$

- $\forall C_i \in C^{ext}, Inv(l_i) = \bigwedge_{x\in\chi_i, t\in trans_i(x)}(x \leq \beta(t))$.
- $\forall C \in C^{ext}, \forall x \in \chi, Dif(C)(x) = 1$ *si*
  $\forall t \in trans(x), t$ *est active*, $Dif(C)(x) = 0$, *sinon.*

7.3 Exemple

La figure 7.2 présente un réseau de Petri T-temporel étendu à l'ordonnancement modélisant deux tâches périodiques s'exécutant en concurrence sur le même processeur et synchronisées par un sémaphore. La figure 7.3 montre l'automate à chronomètres des classes d'états correspondant.

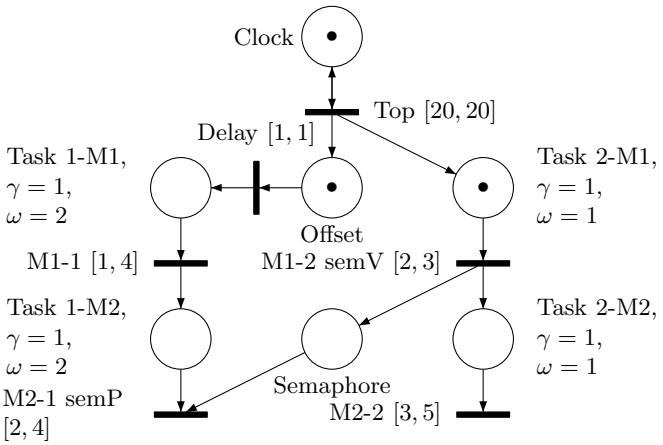

FIG. 7.2 – *Scheduling*-TPN modélisant deux tâches sur un même processeur synchronisées par un sémaphore.

7.4 Propriétés

7.4.1 Bisimulation

Lorsque le calcul se termine, l'automate à chronomètres produit et le réseau de Petri T-temporel étendu à l'ordonnancement initial sont temporellement bisimilaires, ce qui prouve la correction de la traduction.

Théorème 20 (Bisimulation) *Soient $Q_{\mathcal{T}}$ l'ensemble des états du réseau de Petri T-temporel $\mathcal{T}$ et $Q_{\mathcal{A}}$ l'ensemble des états de son automate à chronomètres des classes d'états $\mathcal{A} = (L, l_0, X, A, E, Inv, Dif)$. Soit $\mathcal{R} \subset Q_{\mathcal{T}} \times Q_{\mathcal{A}}$ une relation binaire telle que $\forall s = (M_{\mathcal{T}}, \nu_{\mathcal{T}}) \in Q_{\mathcal{T}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}, s\mathcal{R}a \Leftrightarrow M_{\mathcal{T}} = M_{\mathcal{A}}$ si $M_{\mathcal{A}}$ est le marquage de la classe d'états étendue l et $\forall t \in enabled(M_{\mathcal{T}}), \exists x \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x)$ si $\dot{x} = 1$ si t est active et $\dot{x} = 0$ sinon.*

$\mathcal{R}$ est une bisimulation.

Preuve. Supposons que $s = (M_{\mathcal{T}}, \nu_{\mathcal{T}}) \in Q_{\mathcal{T}}, a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}$ et $s\mathcal{R}a$. Alors $\forall t \in enabled(M_{\mathcal{T}}), \exists x_t \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x_t)$.

1. Supposons que le Scheduling-TPN puisse laisser s'écouler le temps $d \in \mathbb{R}^+ : s \xrightarrow{d} s'$. Cela signifie que $\forall t \in enabled(M_{\mathcal{T}})$, $\nu_{\mathcal{T}}(t) + d \leq \beta(t)$. Donc, $\nu_{\mathcal{A}}(x_t) + d \leq \beta(t)$ et donc par définition de $Inv(l)$, $Inv(l)(\nu_{\mathcal{A}} + d')$ est vrai $\forall d' \leq d$. Par conséquent, le SWA $\mathcal{A}$ peut laisser le temps d s'écouler : $a \xrightarrow{d} a'$. Puisque le Scheduling-TPN reste dans le même marquage, et le SWA dans la même localité, l'activité des transitions et les conditions sur $\dot{x}$ ne changent pas. Par conséquent, $\nu_{\mathcal{T}} + d = \nu_{\mathcal{A}} + d$ et finalement $s'\mathcal{R}a'$.
2. Supposons que le Scheduling-TPN puisse tirer la transition $t \in T : s \xrightarrow{t} s'$. Par définition de l'automate à chronomètres des classes d'états, il existe une arête $e = (l, \delta, t, R, \rho, l')$. Cela signifie que $\alpha(t) \leq \nu_{\mathcal{T}}(t)$. Donc, $\alpha(t) \leq \nu_{\mathcal{A}}(x_t)$ et par définition de la garde δ, $\delta(\nu_{\mathcal{A}})$ est vérifiée. Donc le SWA $\mathcal{A}$ peut prendre l'arête $e : a \xrightarrow{t} a'$. Par définition de l', le marquage $M'_{\mathcal{A}}$ de la classe d'états étendue l' est le même que le nouveau marquage $M'_{\mathcal{T}}$ du Scheduling-TPN. Soit t' une transition de $enabled(M'_{\mathcal{T}})$.

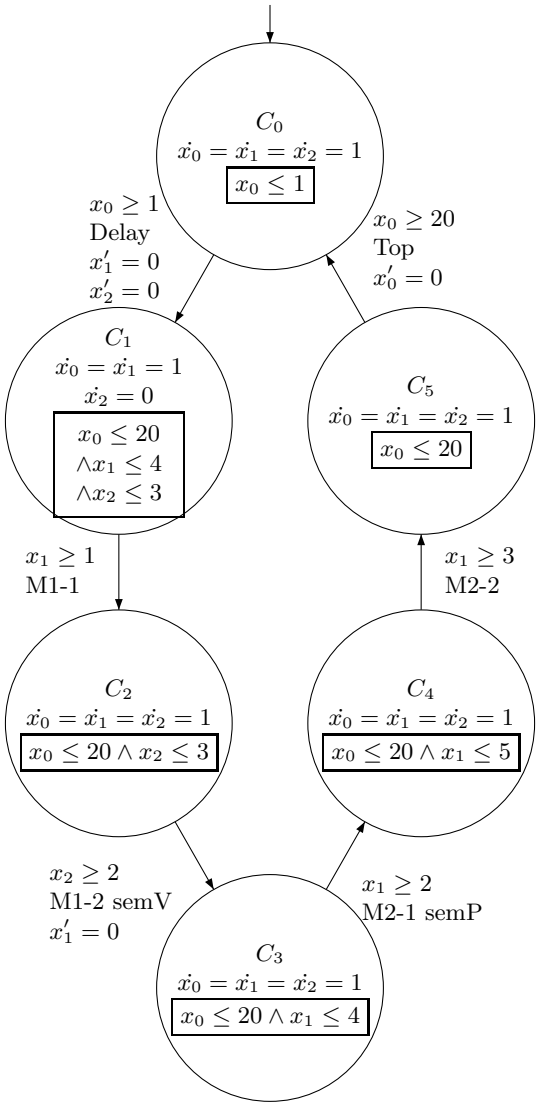

FIG. 7.3 – Automate à chronomètres des classes d'états du réseau de la figure 7.2. La localité initiale est C_0.

Si t' est nouvellement sensibilisée, un nouveau chronomètre est créé ou t' est associée à un chronomètre dont la valeur est 0 et dont l'activité est la même que celle de t'. Dans le premier cas, l'activité du chronomètre est également définie en accord avec l'activité de t'. Si t' n'est pas nouvellement sensibilisée, et si toutes les transitions associées à son chronomètre ont la même activité, alors l'activité du chronomètre est redéfinie en accord avec l'activité de t', sinon, si toutes les transitions associées à ce chronomètre n'ont pas la même activité, un nouveau chronomètre est créé dont l'activité est celle de t' et auquel t' est associée. Finalement, $s'\mathcal{R}a'$.

3. Supposons que le SWA puisse laisser le temps $d \in \mathbb{R}^+$ s'écouler : $a \xrightarrow{d} a'$. Cela signifie que $Inv(l)(\nu_\mathcal{A} + d)$ est vrai. Donc, par définition de $Inv(l)$, $\forall x \in X, \forall t \in trans(x)$, $\nu_\mathcal{A}(x)+d \leq \beta(t)$, ce qui est équivalent à $\forall x \in X, \forall t \in trans(x), \nu_\mathcal{T}(t)+d \leq \beta(t)$. Puisque $\bigcup_{x\in X} trans(x) = enabled(M_\mathcal{T})$, nous avons finalement, $\forall t \in enabled(M_\mathcal{T}), \nu_\mathcal{T}(t) + d \leq \beta(t)$, ce qui signifie que $\mathcal{T}$ peut laisser le temps d s'écouler : $s \xrightarrow{d} s'$. Comme au premier point, $\nu_\mathcal{T} + d = \nu_\mathcal{A} + d$ et donc, $s'\mathcal{R}a'$.

4. Supposons que le SWA puisse prendre l'arête $e = (l, \delta, t, R, \rho, l')$: $a \xrightarrow{t} a'$. Cela signifie que t est sensibilisée par $M_\mathcal{A} = M_\mathcal{T}$ et que $\delta(\nu_\mathcal{A})$ est vraie. Donc, par définition de δ, $\nu_\mathcal{A}(x_t) \geq \alpha(t)$ et donc $\nu_\mathcal{T}(t) \geq \alpha(t)$, ce qui signifie que t est tirable pour $\mathcal{T}$: $s \xrightarrow{t} s'$. Comme au deuxième point, $s'\mathcal{R}a'$ par construction.

□

7.4.2 Surapproximation

Nous avons montré au chapitre 6 que le domaine des classes d'états peut être surapproximé par une DBM. Nous pouvons bien sûr faire la même chose pour les classes d'états étendues. Nous obtenons alors un automate à chronomètres dont le comportement contient celui du réseau de Petri T-temporel étendu à l'ordonnancement. En fait nous allons montrer que cet automate est toujours temporellement bisimilaire au réseau de Petri. En effet, puisque les gardes et invariants sont calculés de manière « statique », les états supplémentaires ajoutés par la surapproximation ne sont pas accessibles :

Théorème 21 (Bisimulation) *Soit $Q_\mathcal{T}$ l'ensemble des états du réseau de Petri T-temporel $\mathcal{T}$ et $Q_\mathcal{A}$ l'ensemble des états de son automate à chronomètres des classes d'états* surapproximé $\mathcal{A} = (L, l_0, X, A, E, Inv, Dif)$. *Soit $\mathcal{R} \subset Q_\mathcal{T} \times Q_\mathcal{A}$ une relation binaire telle que $\forall s = (M_\mathcal{T}, \nu_\mathcal{T}) \in Q_\mathcal{T}, \forall a = (l, \nu_\mathcal{A}) \in Q_\mathcal{A}, s\mathcal{R}a \Leftrightarrow M_\mathcal{T} = M_\mathcal{A}$ si $M_\mathcal{A}$ est le marquage de la classe d'états étendue l et $\forall t \in enabled(M_\mathcal{T}), \exists x \in X, \nu_\mathcal{T}(t) = \nu_\mathcal{A}(x)$ si $\dot{x} = 1$ si t est active et $\dot{x} = 0$ sinon.*

$\mathcal{R}$ est une bisimulation.

La bisimulation est la même que pour l'automate exact et la preuve également. Pour s'en convaincre complètement, supposons qu'il existe dans la localité l une arête sortante $e = (l, \delta, t, R, \rho, l')$ car t est tirable dans la classe d'états *surapproximée* l alors qu'elle ne l'est pas dans la classe *exacte* correspondante. Si nous supposons qu'avant d'atteindre la localité l, le comportement de l'automate était correct (exact), alors à l'instant précis de l'entrée dans la classe l, l'automate est dans un état $a = (M, \nu_\mathcal{A})$ qui est en relation avec un état $s = (M, \nu_\mathcal{T})$ du Scheduling-TPN $\mathcal{R}$. D'une part, puisque t est en fait non tirable, cela signifie qu'il existe une autre transition t' qui doit être tirée avant elle : $\alpha(t) - \nu_\mathcal{T}(t) > \beta(t') - \nu_\mathcal{T}(t')$. Donc, par

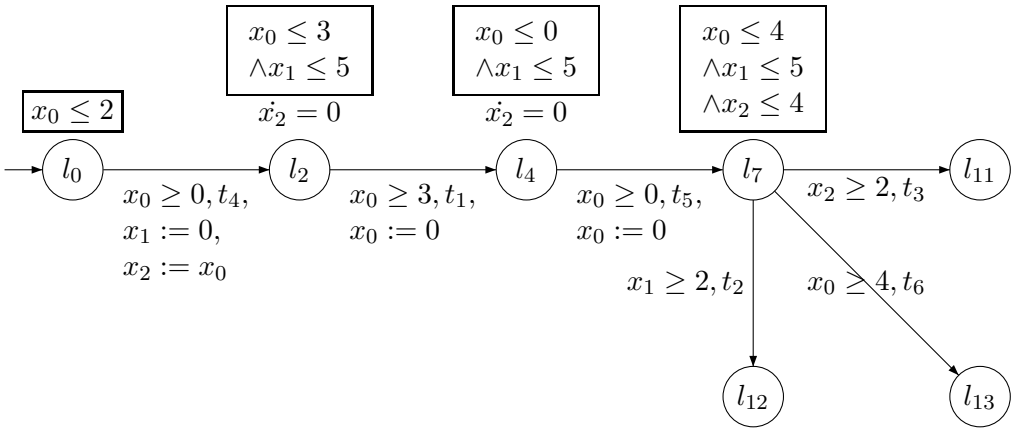

FIG. 7.4 – Traduction partielle en SWA du réseau de la figure 6.17

définition de $\mathcal{R}$, il existe un chronomètre x_t tel que $\alpha(t) - \nu_{\mathcal{A}}(x_t) > \beta(t') - \nu_{\mathcal{T}}(t')$. Puisque, par définition d'un *Scheduling*-TPN, $\beta(t') \geq \nu_{\mathcal{T}}(t')$, cela nous donne $\alpha(t) - \nu_{\mathcal{A}}(x_t) > 0$.

D'autre part, par définition des gardes de l'automate à chronomètres des classes d'états, la garde δ est vraie est équivalent à $\alpha(t) - \nu_{\mathcal{A}}(x_t) \leq 0$. Avec ce qui précède, nous pouvons donc conclure que la garde δ est fausse et donc l' n'est pas accessible.

Cela est illustré par la figure 7.4 qui présente la partie de l'automate à chronomètres des classes d'états du réseau de la figure 6.17 du chapitre 6 correspondant à la séquence à laquelle nous nous étions intéressés. Nous voyons que la localité l_{13} correspondant au tir « parasite » de la transition t_6 n'est en fait pas accessible. En effet, à l'entrée dans la localité l_7, nous avons les équations suivantes $x_0 = 0$, $x_1 = 3 - \tau$ et $x_2 = \tau$, en notant τ l'instant de tir de t_4 ($0 \leq \tau \leq 2$). Par conséquent, nous pouvons laisser s'écouler $\min\{4, 2 + \tau, 4 - \tau\}$ unités de temps et il est clair que $\forall \tau \in]0, 2], 4 - \tau < 4$ et $\forall \tau \in [0, 2[, 2 + \tau < 4$. Donc la garde $x_0 \geq 4$ ne sera jamais vérifiée.

7.5 Vérification

Pour vérifier l'automate à chronomètres des classes d'états nous pouvons utiliser HYTECH. Cet outil permet l'analyse symbolique des automates hybrides linéaires et est basé sur la manipulation symbolique de « régions » décrites par des polyèdres. HYTECH fournit un certain nombre d'opérations pour manipuler les régions, incluant le calcul de l'ensemble des états accessibles depuis une région, le calcul des successeurs, la quantification existentielle, l'enveloppe convexe et les opérations élémentaires booléennes (comparaison, test du vide,...). Nous verrons en détail au chapitre 10 comment faire la vérification en pratique.

Nous avons comparé l'efficacité de notre méthode, implémentée dans l'outil ROMEO[5], avec une modélisation générique directe en automate à chronomètres. La modélisation directe est

[5]voir le chapitre 9

générique et se présente sous la forme d'un produit synchronisé d'automates à chronomètres.

7.5.1 Modélisation directe

Cette modélisation requiert un automate par tâche et un par ordonnanceur (et donc par processeur).

La figure 7.5 donne l'automate générique modélisant une tâche τ_i périodique de période T_i et dont la durée d'exécution appartient à l'intervalle $[\alpha_i, \beta_i]$. Cette modélisation s'inspire du modèle des tâches étendues d'OSEK/VDX [gro01]. Elle comporte notamment un état $Waiting$ pour l'attente de message. Les mécanismes de synchronisation fournissant les événements $release_i!$ et $waitEvent_i!$ et réagissant aux événements $sendEvent_i!$ sont modélisés à part, sous la forme d'automates supplémentaires. Ce modèle de tâche utilise une horloge o_i pour l'activation périodique et un chronomètre x_i, en tant que compteur de temps d'exécution.

La figure 7.6 présente le modèle de l'ordonnanceur à priorités fixes. À chaque événement nécessitant un réordonnancement, toutes les tâches reçoivent un signal de préemption puis la plus prioritaire est démarrée. Les priorités sont implicites et découlent de l'ordre dans lequel l'ordonnanceur tente de démarrer les tâches. La structure de l'ordonnanceur dépend du nombre n de tâches et la localité $Scheduling$ doit être dupliquée $n-1$ fois . Nous utilisons également une horloge u pour simuler l'urgence dans les localités de l'ordonnanceur, à l'exception de $Idle$.

Quand cela était possible, nous avons modélisé les activations périodiques des tâches de même période par un seul et même chronomètre ce qui crée, par contre, un automate supplémentaire.

7.5.2 Résultats

Nous avons comparé notre méthode et la modélisation directe sur un ensemble de systèmes de complexité croissante : de deux processeurs avec quatre tâches à sept processeurs reliés par un bus CAN avec dix-huit tâches. Le tableau 7.1 donne les résultats obtenus.

Les colonnes 2 et 3 donnent le nombre de processeurs et de tâches du système. Les colonnes 4, 5 et 6 décrivent les résultats obtenus avec la modélisation directe en automates à chronomètres : le nombre d'automates à chronomètres et le nombre de chronomètres utilisés pour la modélisation et le temps mis par HYTECH pour calculer l'espace d'états du système. Les colonnes 7, 8 et 9 donnent les résultats pour notre méthode. Nous donnons le nombre de localités et de transitions de l'automate à chronomètres calculé par notre semi-algorithme ainsi que le temps mis pour ce calcul. Enfin la dernière colonne donne le temps mis par HYTECH pour calculer l'espace d'états de cet automate. Les temps sont donnés en secondes et NA indique que le calcul n'a pas abouti avec HYTECH sur la machine utilisée. Celle-ci est à base POWERPC G4 à 1.25GHz avec 500Mo de RAM.

Pour comparer l'efficacité des deux approches, nous pouvons comparer la somme des deux dernières colonnes avec le nombre de la colonne 6. Nous constatons que, très vite, le calcul de l'espace d'états du modèle issu de la modélisation générique directe sous la forme d'un produit d'automates à chronomètres devient impossible alors que la complexité du système augmente. Notre méthode, par contre, donne de bons résultats jusqu'à ce que l'automate qu'elle produit ne soit plus analysable non plus par HYTECH (dernière ligne). Dans ce cas, pour des propriétés de sûreté, nous pouvons toujours utiliser le graphe des classes étendues

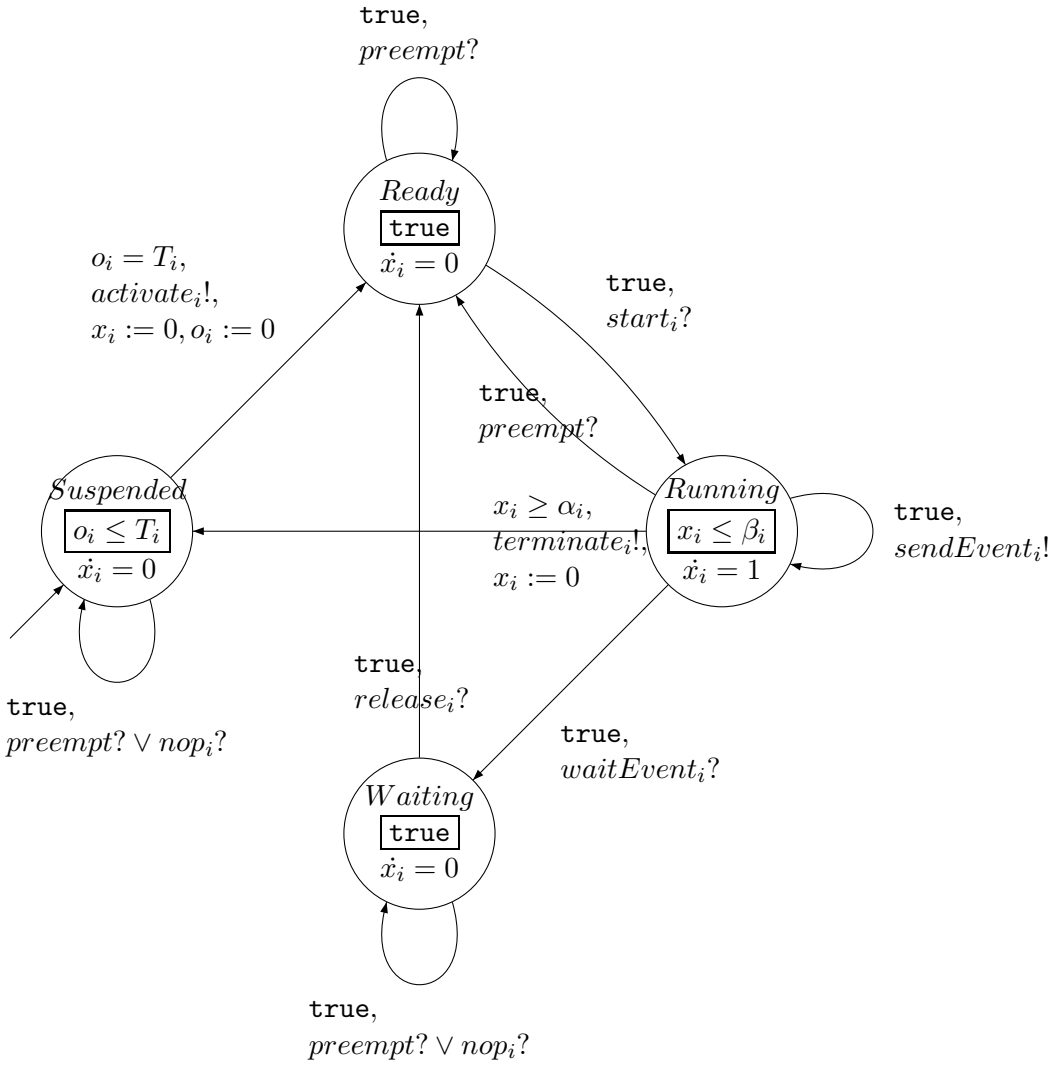

FIG. 7.5 – Automate à chronomètres d'une tâche temps réel

généré par notre semi-algorithme à base de DBM, en gardant à l'esprit que nous sommes alors en présence d'une surapproximation de l'espace d'états.

7.6 Conclusion

Dans ce chapitre, nous avons présenté une méthode pour l'analyse efficace des systèmes temps réel. Elle consiste dans un premier temps à modéliser le système sous la forme d'un réseau de Petri T-temporel étendu à l'ordonnancement. Dans un second temps, nous appliquons un semi-algorithme de traduction en automate à chronomètres. Ce semi-algorithme présente deux avantages principaux. Premièrement, il est rapide car il calcule des suraproximations de chaque classe d'états sous la forme de DBM. L'automate à chronomètres résultat est néanmoins temporellement bisimilaire au réseau de Petri T-temporel étendu à l'ordonnancement initial car les localités supplémentaires de l'automate à chronomètres, générées

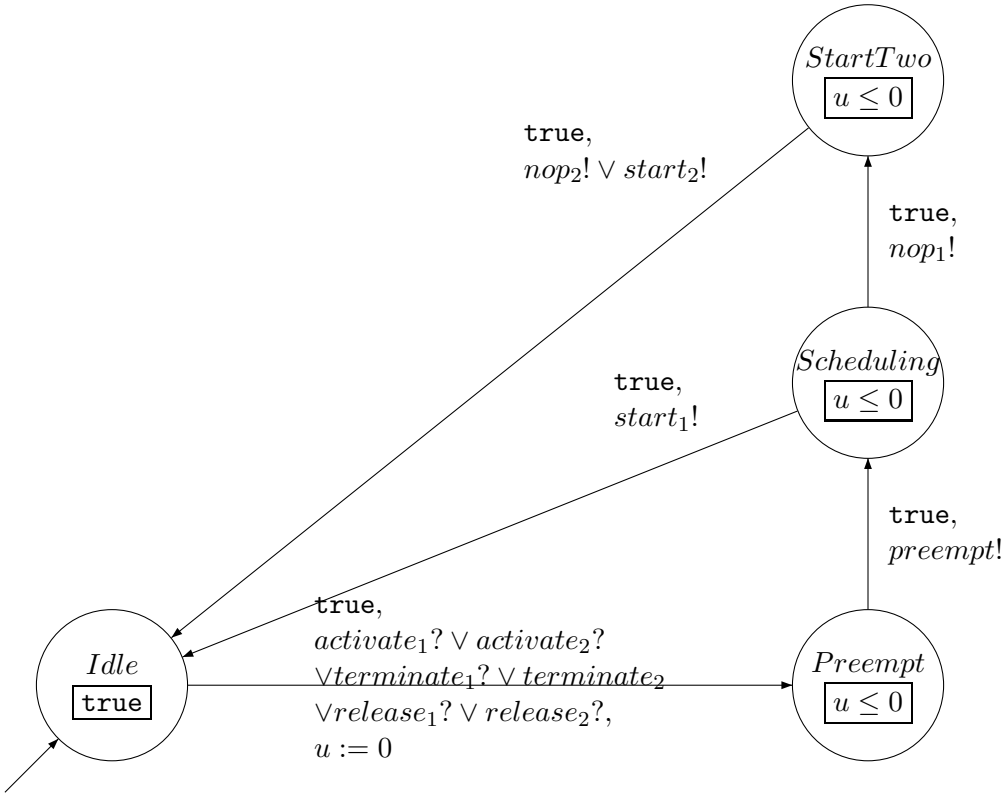

FIG. 7.6 – Automate à chronomètres d'un ordonnanceur à priorités fixes pour deux tâches

éventuellement par l'approximation ne sont pas accessibles. Deuxièmement, il fournit un automate dont le nombre de chronomètres est minimisé par des techniques de réduction à la volée. La vérification de cet automate est donc plus efficace. Nous avons montré que, dans la pratique, cette approche est plus efficace qu'une modélisation générique directe du système sous la forme d'un produit synchronisé d'automates à chronomètres. Elle permet notamment de vérifier des systèmes bien plus importants en termes de nombre de processeurs et de tâches.

	Description		Modélisation directe (SWA)			Notre méthode (*Scheduling*-TPN $\overset{\text{Romeo}}{\longrightarrow}$ SWA $\overset{\text{HyTech}}{\longrightarrow}$ state-space)				
Ex.	Proc.	Tâches	SWA's	Sw.	Temps HyTech	Localités	Transitions	Sw.	Temps Romeo	Temps HyTech
1	2	4	8	7	77.8	20	29	3	$\leq$0.1	0.2
2	3	6	11	9	590.3	40	58	4	$\leq$0.1	0.5
3	3	7	12	10	NA	52	84	4	$\leq$0.1	0.7
4	3+CAN	7	13	11	NA	297	575	7	0.3	5.3
5	4+CAN	9	15	13	NA	761	1677	8	0.9	29.8
6	5+CAN	11	17	15	NA	1141	2626	9	6	60.1
7	5+CAN	12	18	16	NA	2155	5576	9	8.3	56.5
8	6+CAN	14	.	.	NA	4587	12777	10	59.7	438.8
9	6+CAN	15	.	.	NA	4868	13155	11	96.5	1364.3
10	6+CAN	16	.	.	NA	5672	15102	11	439.1	1372.5
11	7+CAN	18	.	.	NA	8817	25874	12	1146.7	NA

Tab. 7.1 – Résultats expérimentaux

Chapitre 8

Extension à d'autres politiques d'ordonnancement

Dans ce chapitre, nous explorons les possibilités d'extension du modèle de [RD02] pour la prise en compte des problèmes d'ordonnancement plus complexes. Dans un premier temps, nous permettons aux vitesses d'évolution des valuations des transitions (c'est-à-dire leurs dérivées par rapport au temps) de prendre des valeurs rationnelles arbitraires, en particulier autres que 0 et 1. Cela nous permet de modéliser la politique d'ordonnancement *Round-Robin* par une approche *fluide*.

Dans un second temps, nous considérons que l'ordonnancement n'est plus uniquement fonction de l'état discret du réseau (le marquage) mais également, dans une certaine mesure, des valuations des transitions. Cela nous permet de modéliser la politique d'ordonnancement *Earliest Deadline First*.

Pour ces extensions, nous ne pouvons plus baser notre modèle sur un retour d'état de la forme $Act(M)$. Nous allons donc définir directement les flux d'évolution des transitions (en fait, des horloges associées aux transitions).

Dans les deux cas, nous proposons une extension de la méthode de calcul du graphe des classes d'états. Dans le cas d'*Earliest Deadline First*, cela implique un partitionnement plus fin que les classes d'états permettant de garantir que tous les états d'une même partition définissent les mêmes dynamiques d'évolution des valuations des transitions.

8.1 Round-Robin

8.1.1 Problématique

Une des hypothèses faite pour la modélisation des politiques d'ordonnancement à priorités fixes est que deux places affectées au même processeur et de même priorité ne peuvent pas être marquées simultanément si elles sensibilisent alors chacune au moins une transition. Cela permet d'obtenir un marquage actif unique. En effet, le processeur ne pouvant exécuter qu'une seule tâche en même temps, il faut choisir quelle tâche est exécutée et donc, pour notre modèle, quelle place nous devons marquer *active*. Pour lever cette hypothèse, nous pouvons faire un choix arbitraire. Par exemple, de toujours marquer active la place d'indice le plus petit. Ou le choix peut être aléatoire et dans ce cas, lors de la vérification, il faut examiner toutes les possibilités.

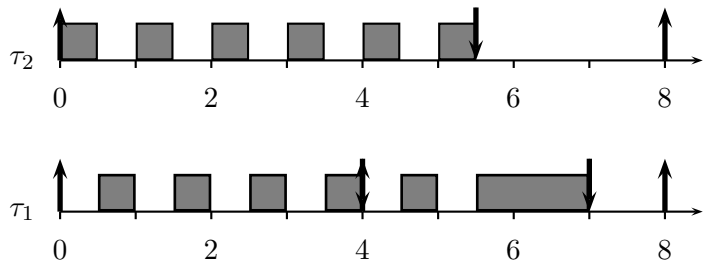

FIG. 8.1 – Chronogramme de l'exécution de deux tâches de même priorité ordonnancées sur le même processeur en *Round-Robin*

Cependant, quand n tâches de même priorité, en concurrence sur le même processeur, se retrouvent prêtes simultanément, une politique couramment adoptée consiste à faire du *pseudo parallélisme*, c'est-à-dire à partager le temps du processeur de façon égale entre chaque tâche en affectant à chacune d'elle, alternativement, de petites « tranches » de temps de même durée d.

La figure 8.1 présente l'exemple de deux tâches τ_1 et τ_2 en concurrence sur le même processeur et de même priorité. La tâche τ_1 est périodique de période 4. Son temps d'exécution est 2 unités de temps. La tâche τ_2 est périodique de période 8 et son temps d'exécution est 3. Enfin, $d = 0.5$.

En faisant tendre d vers 0, nous pouvons modéliser cette politique en supposant que le processeur se partage parfaitement bien entre les n tâches : la progression de chacune des tâches, et donc des m transitions sensibilisées par les n places de même priorité, s'effectue à la vitesse $1/m$. Cela est illustré par la figure 8.2. Plus la granularité est petite devant le temps d'exécution des tâches, plus le modèle est proche de la réalité. C'est cette approche *fluide* que nous allons adopter.

Bien sûr, pour modéliser cela nous ne pouvons pas utiliser le modèle du chapitre 6 tel quel. Nous allons donc l'étendre.

8.1.2 Définitions

Définition 38 (*Scheduling*-TPN) *Un réseau de Petri T-temporel étendu à l'ordonnancement est un 8-uplet $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0, Flow)$, où*

- $P = \{p_1, p_2, \ldots, p_m\}$ *est un ensemble fini et non vide de* places *;*
- $T = \{t_1, t_2, \ldots, t_n\}$ *est un ensemble fini et non vide de* transitions *($T \cap P = \emptyset$) ;*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *est la* fonction d'incidence amont *;*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *est la* fonction d'incidence aval *;*
- $M_0 \in \mathbb{N}^P$ *est le* marquage initial *du réseau ;*
- $\alpha \in (\mathbb{R}^+)^T$ *et* $\beta \in (\mathbb{R}^+ \cup \{\infty\})^T$ *sont les fonctions donnant pour chaque transition, respectivement son instant de tir* au plus tôt *et* au plus tard *($\alpha \leq \beta$).*
- $Flow \in (\mathbb{N}^P)^{T^{\mathbb{R}+}}$ *est la* fonction d'activité.

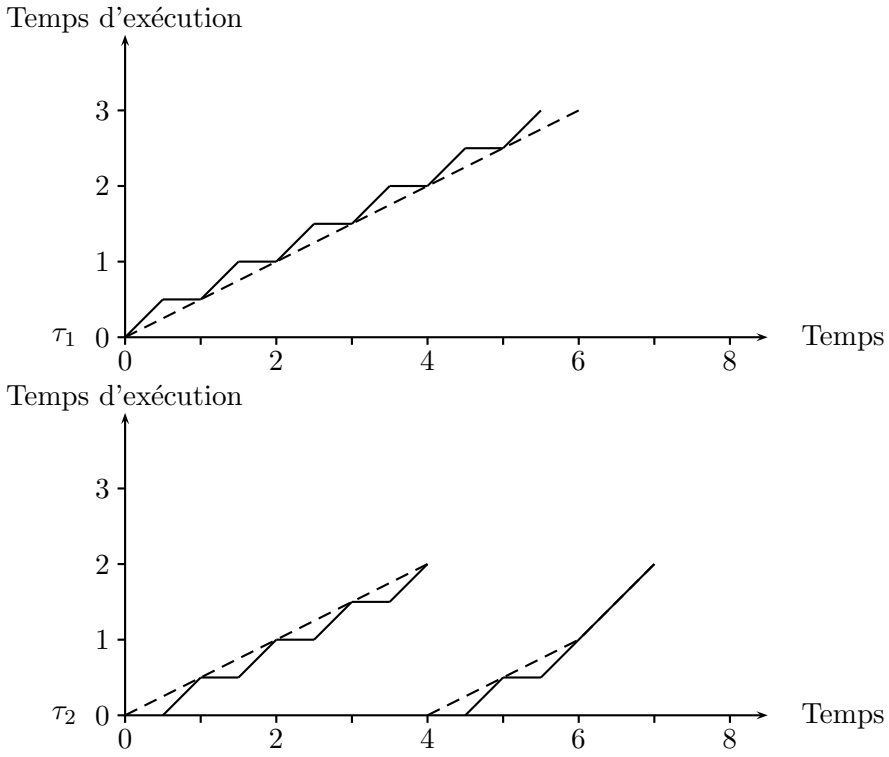

FIG. 8.2 – Temps d'exécution des deux tâches de la figure 8.1 en fonction du temps : réel (trait plein) et modèle (trait interrompu)

Une transition t est dite *active* pour un marquage M si t est sensibilisée et $Flow(M)(t) \neq 0$.

Définition 39 (Sémantique d'un *Scheduling*-TPN) *La sémantique d'un réseau de Petri T-temporel étendu à l'ordonnancement $\mathcal{T}$ est définie sous la forme d'un système de transitions temporisé $S_{\mathcal{T}} = (Q, q_0, \rightarrow)$ tel que :*

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$;
- $q_0 = (M_0, \overline{0})$;
- $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ *est la relation de transition incluant des transitions continues et des transitions discrètes :*
  - *la relation de transition* continue *est définie* $\forall d \in \mathbb{R}^+$ *par :*

$$(M, \nu) \xrightarrow{d} (M, \nu') \text{ ssi } \begin{cases} \forall t_i \in enabled(M), \nu'(t_i) = \nu(t_i) + Flow(M)(t_i) * d, \\ \forall t_k \in T, M \geq {}^\bullet t_k \Rightarrow \nu'(t_k) \leq \beta(t_k). \end{cases}$$

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

$$(M, \nu) \xrightarrow{t_i} (M', \nu') \text{ ssi } \begin{cases} t_i \in enabled(M) \wedge Flow(M)(t_i) \neq 0, \\ \alpha(t_i) \leq \nu(t_i) \leq \beta(t_i), \\ M' = M - {}^\bullet t_i + t_i^\bullet, \\ \forall t_k, \nu'(t_k) = \begin{cases} 0 \text{ si } \uparrow enabled(t_k, M, t_i), \\ \nu(t_k) \text{ sinon.} \end{cases} \end{cases}$$

La définition de $\uparrow enabled$ reste la même que pour le modèle du chapitre 6.

Nous pouvons retrouver facilement la définition du chapitre 6 avec $\forall t \in enabled(M), Flow(M)(t) = 1$ si $t \in enabled(Act'(M))$ et $Flow(M)(t) = 0$ sinon.

Bien que plus générale, cette définition ne change pas la complexité de la vérification par rapport au modèle de [RD02]. Comme pour les automates hybrides linéaires temporellement déterministes [ACH+95], la vitesse d'évolution des variables continues est ici déterminée par l'état discret du modèle.

8.1.3 Calcul de la fonction $Flow$

La fonction $Flow$, comme la fonction Act du chapitre 6 dépend de la politique d'ordonnancement choisie.

Dans le cas particulier d'une politique à priorités fixes, nous pouvons utiliser les paramètres γ et ω introduits au chapitre 6. Par ailleurs, nous imposons que toute transition ait au plus une place amont p affectée à un processeur (telle que $\gamma(p) \neq \phi$). Soit un marquage M, la fonction $Flow$ peut alors être définie de la façon suivante :

1. déterminer l'ensemble $\mathcal{P}$ des places marquées par M qui sensibilisent au moins une transition ;
2. pour tout p de $\mathcal{P}$, telle que $\gamma(p) \neq \phi$, si t est une transition sensibilisée par p alors :
   - si p a une priorité supérieure ou égale à celle des autres places de $\mathcal{P}$ associées au même processeur, alors $Flow(M)(t) = \frac{1}{n}$, où n est le nombre de transitions sensibilisées par des places de $\mathcal{P}$ qui ont la même priorité sur le même processeur que p (y compris t),
   - sinon $Flow(M)(t) = 0$.

La Figure 8.3 présente le modèle d'un système de quatre tâches τ_1, τ_2, τ_3 et τ_4 en concurrence sur le même processeur. τ_1, τ_2 et τ_3 ont la même priorité. τ_4 a une priorité inférieure. Par conséquent, nous avons, dans l'état initial $(M_0, \overline{0})$: $Flow(M_0)(t_1) = Flow(M_0)(t_2) = Flow(M_0)(t_3) = \frac{1}{3}$ et $Flow(M_0)(t_4) = 0$.

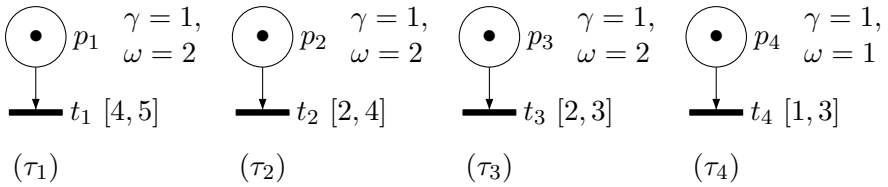

FIG. 8.3 – Un réseau de Petri T-temporel étendu à l'ordonnancement

8.1.4 Successeur d'une classe d'états

Avec la nouvelle définition de l'activité d'une transition, nous avons la définition suivante :

Définition 40 (Transition tirable depuis une classe) *Soit $C = (M, D)$ une classe d'états d'un réseau de Petri T-temporel étendu à l'ordonnancement. Une transition t_i est dite* tirable *depuis C ssi t_i est active et il existe une solution $(\theta_1^*, \ldots, \theta_n^*)$ de D telle que $\forall j \in [\![1, n]\!] - \{i\}$, t.q. t_j est active , $\frac{\theta_i^*}{Flow(M)(t_i)} \leq \frac{\theta_j^*}{Flow(M)(t_j)}$.*

Le successeur d'une classe est calculé de la façon suivante :

Soient une classe $C = (M, D)$ et une transition tirable t_f, la classe $C' = (M', D')$, successeur de C par t_f, est donnée par :

- Le nouveau marquage est calculé de façon classique par $M' = M - {}^\bullet t_f + t_f{}^\bullet$
- Le nouveau domaine de tir, D', est calculé selon les étapes suivantes (nous le noterons $next(D, t_f)$) :

  1. changements de variables $\forall j$, t.q. t_j est sensibilisée, $\theta_j = \frac{Flow(M)(t_j)}{Flow(M)(t_f)}\theta_f + \theta'_j$;
  2. $\forall j \neq t_f$, ajout des contraintes $\theta'_j \geq 0$;
  3. élimination des variables correspondant à des transitions désensibilisées par le tir de t_f (ce qui inclut donc t_f), par exemple en utilisant la méthode de Fourier-Motzkin [Dan63] ;
  4. ajout des inéquations relatives aux transitions nouvellement sensibilisées par le tir de t_f :
  $$\forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq \theta'_k \leq \beta(t_k).$$
  5. détermination de la forme canonique D'^* du nouveau domaine de tir.

Le seul changement, par rapport au modèle du chapitre 6, réside dans le changement de variables car les vitesses d'évolution des valuations des transitions sont différentes. Soient deux transitions t_1 et t_2 sensibilisées par le marquage M et tirées consécutivement. Leurs vitesses d'évolution sont respectivement $Flow(M)(t_1)$ et $Flow(M)(t_2)$. Le temps jusqu'au tir de t_2 est égal au temps jusqu'au tir de t_1 plus le temps entre le tir de t_1 et le tir de t_2. Ce qui s'écrit :

$$\frac{\theta_2}{Flow(M)(t_2)} = \frac{\theta_1}{Flow(M)(t_1)} + \frac{\theta'_2}{Flow(M)(t_2)}$$

Ou encore :

$$\theta_2 = \frac{Flow(M)(t_2)}{Flow(M)(t_1)}\theta_1 + \theta'_2$$

FIG. 8.4 – Un réseau de Petri T-temporel étendu à l'ordonnancement

8.1.5 Exemple

Le réseau de la figure 8.4 représente le système donné en introduction et dont le chronogramme de l'exécution est représenté sur la figure 8.1.

Nous allons calculer son graphe des classes d'états. Le résultat est donné sur la figure 8.5. Les inéquations redondantes dans les domaines de tir seront omises.

La classe initiale est

$$C_0 = \begin{cases} \{p_1, p_2, p_3, p_4\} \\ 4 \leq \theta_1 \leq 4 \\ 8 \leq \theta_3 \leq 8 \\ 2 \leq \theta_2 \leq 2 \\ 3 \leq \theta_4 \leq 3 \end{cases}$$

t_2 et t_4 sont affectées au même processeur avec la même priorité. Elles sont donc actives toutes les deux mais le processeur est partagé idéalement entre elles : $Flow(M_0)(t_2) = Flow(M_0)(t_4) = \frac{1}{2}$. t_1 et t_2 sont tirables. Nous tirons t_2. Le changement de variable se fait comme suit :

$$C_1' = \begin{cases} \{p_1, p_3, p_4\} \\ 4 \leq \theta_1' + 2\theta_2 \leq 4 \\ 8 \leq \theta_3' + 2\theta_2 \leq 8 \\ 2 \leq \theta_2 \leq 2 \\ 3 \leq \theta_4 + \frac{1}{2}2\theta_2 \leq 3 \end{cases}$$

Et donc la classe obtenue est finalement :

$$C_1 = \begin{cases} \{p_1, p_3, p_4\} \\ 0 \leq \theta_1 \leq 0 \\ 4 \leq \theta_3 \leq 4 \\ 1 \leq \theta_4 \leq 1 \end{cases}$$

Puis, t_1 est tirée en 0 unités de temps, ce qui nous donne la classe suivante :

$$C_2 = \begin{cases} \{p_1, p_2, p_3, p_4\} \\ 4 \leq \theta_1 \leq 4 \\ 4 \leq \theta_3 \leq 4 \\ 2 \leq \theta_2 \leq 2 \\ 1 \leq \theta_4 \leq 1 \end{cases}$$

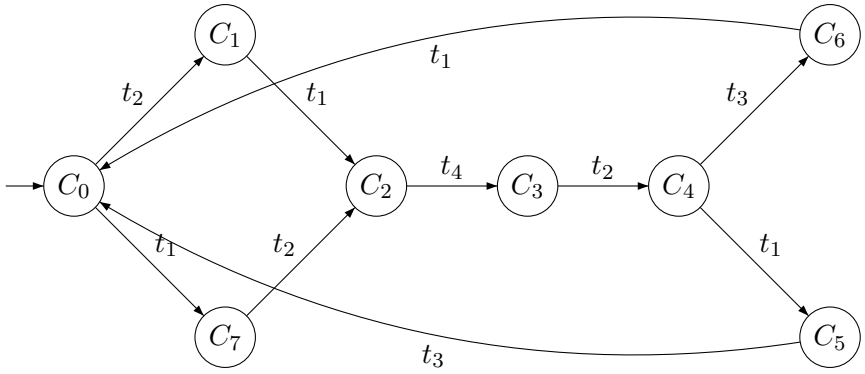

FIG. 8.5 – Graphe des classes d'états du réseau de la figure 8.4

Le tir de t_1 puis t_2 conduit à cette même classe. Seule t_4 est tirable. A nouveau, t_2 et t_4 sont en concurrence sur le même processeur avec la même priorité. Nous avons donc :

$$C'_3 = \begin{cases} \{p_1, p_2, p_3\} \\ 4 \leq \theta'_1 + 2\theta_4 \leq 4 \\ 4 \leq \theta'_3 + 2\theta_4 \leq 4 \\ 2 \leq \theta'_2 + \frac{2}{2}\theta_4 \leq 2 \\ 1 \leq \theta_4 \leq 1 \end{cases} \quad C_3 = \begin{cases} \{p_1, p_2, p_3\} \\ 2 \leq \theta_1 \leq 2 \\ 2 \leq \theta_3 \leq 2 \\ 1 \leq \theta_2 \leq 1 \end{cases}$$

Puis t_2 est la seule transition tirable et est seule sur son processeur donc la classe obtenue après son tir est :

$$C_4 = \begin{cases} \{p_1, p_3\} \\ 1 \leq \theta_1 \leq 1 \\ 1 \leq \theta_3 \leq 1 \end{cases}$$

Finalement les tirs consécutifs de t_1 puis t_3 (ou t_3 puis t_1) conduisent à la classe initiale sans difficulté. La figure 8.5 donne le graphe obtenu.

8.1.6 Conclusion

Nous avons proposé une extension du modèle *Scheduling*-TPNpermettant la modélisation de la politique d'ordonnancement *Round-Robin*. Nous avons également proposé une adaptation de la méthode de calcul du graphe des classes d'états pour ce modèle.

Comme pour le modèle du chapitre 6, l'accessibilité, la bornitude et les autres propriétés « intéressantes » sont indécidables pour cette extension. La méthode de calcul proposée est par conséquent un semi-algorithme.

8.2 Earliest Deadline First

Earliest Deadline First (EDF) [LL73] est une politique d'ordonnancement à priorités dynamiques. Son principe est de donner la plus grande priorité à la tâche dont l'échéance est la plus proche dans le temps. Cette politique a été prouvée *optimale* pour des ensemble de tâches périodiques, indépendantes et à échéances inférieures ou égales à leurs périodes [LL73, Der74].

C'est-à-dire que si un tel ensemble de tâches n'est pas ordonnançable par EDF, alors il n'est ordonnançable par aucune politique à priorités dynamiques.

Étant donné un ensemble de tâches prêtes, l'ordre relatif des échéances de ces tâches est invariant dans le temps ; tout au moins jusqu'à la terminaison de l'une d'elle ou le passage à l'état prêt d'une nouvelle tâche. Du point de vue du modèle réseaux de Petri T-temporels étendus à l'ordonnancement, cela correspond à des changements d'état discret, c'est-à-dire des changements de marquage. C'est cette propriété qui va nous permettre de modéliser cette politique d'ordonnancement car dans notre modèle la *fonction d'activité* (ou le marquage actif) n'est réévaluée qu'aux moments des changements de marquage.

8.2.1 Modélisation

Soient $Tasks$ l'ensemble des tâches du système et $Procs$ l'ensemble des processeurs. Nous notons $\mathcal{P} : Tasks \mapsto Procs$ la fonction qui affecte chaque tâche à un processeur.

Pour définir la fonction d'activité, nous allons associer chaque place du réseau à une tâche par la fonction $\gamma : P \mapsto Tasks \cup \{\phi]$. Comme précédemment ϕ indique que la place n'est affectée à aucune tâche et correspond par exemple à un service de l'exécutif.

Nous imposons que pour toute transition, il y ait *au plus* une place p telle que $p \in {}^\bullet t$ et $\gamma(p) \neq \phi$. Alors pour toute transition t, nous dirons que t *fait partie* de la tâche τ, et nous notons $t \in \tau$, si l'une de ses places amont est associée à τ : $t \in \tau \Leftrightarrow \exists p \in {}^\bullet t$, t.q. $\gamma(p) = \tau$. Par commodité nous notons $\gamma(t)$ la tâche τ telle que $t \in \tau$.

Chaque tâche τ est donc modélisée par un motif *Scheduling*-TPN composé des places affectées à τ et des transitions qui font partie de τ. Nous supposons qu'une seule instance de chaque tâche au plus est active à un instant donné. Cela se traduit par la restriction suivante : au plus une place associée à τ est marquée à un instant donné.

Soit $B(\tau)$ l'ensemble des transitions qui *commencent* la tâche τ. $B(\tau)$ est un ensemble car plusieurs événements différents peuvent être à l'origine de lancement de la tâche τ. De la même façon, nous définissons $E(\tau)$, l'ensemble des transitions qui *finissent* la tâche τ. La définition de ces deux ensembles fait partie de la modélisation.

Enfin, la fonction $\mathcal{D}_s : Tasks \mapsto \mathbb{R}^+$ donne l'échéance des tâches par rapport à leur instant d'activation.

8.2.2 Exemple

Afin d'illustrer cette modélisation, considérons un système de deux tâches τ_1 et τ_2 en concurrence sur un même processeur et ordonnancées par *Earliest Deadline First*.

τ_1 est périodique de période 10. Sa durée d'exécution est comprise entre 4 et 6 unités de temps. À chaque activation, τ_1 déclenche l'activation sporadique de τ_2 après une durée comprise entre 1 et 3 unités de temps. La durée d'exécution de τ_2 est de 2 unités de temps. Enfin les échéances respectives de τ_1 et τ_2 sont de 10 et 8 unités de temps après leurs requêtes d'activation.

les figures 8.6 et 8.7 présentent deux simulations de l'exécution du système lorsque la tâche τ_2 est lancée respectivement après 1 unité de temps et après 3 unités de temps. Les échéances sont matérialisées par les lignes en traits interrompus.

Comme nous l'avons vu au chapitre 2, cette classe de systèmes, avec des précédences, des durées d'exécutions et des dates de réveil variables ne peut être analysée de façon exacte par des méthodes « analytiques » d'ordonnançabilité.

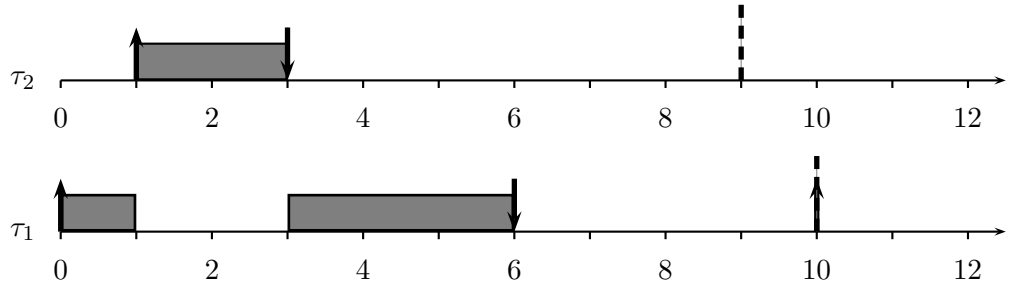

FIG. 8.6 – Chronogramme d'une simulation de l'exécution du système représenté par la figure 8.8 (τ_2 lancée à l'instant 1)

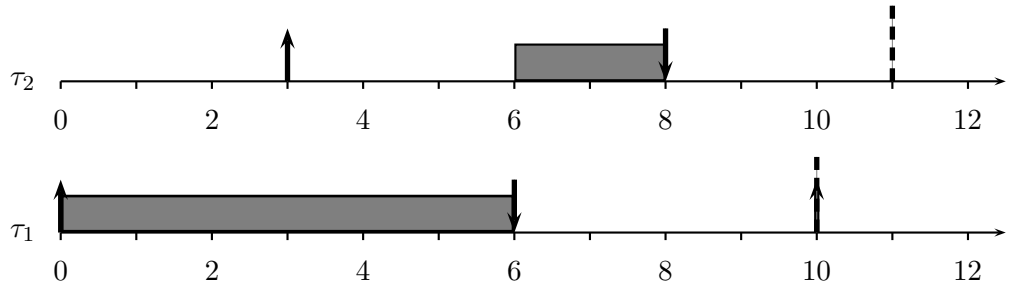

FIG. 8.7 – Chronogramme d'une simulation de l'exécution du système représenté par la figure 8.8 (τ_2 lancée à l'instant 3)

La figure 8.8 donne un exemple de modélisation avec le modèle de *Scheduling*-TPN que nous avons décrit dans la sous-section précédente. p_2 et p_3 sont affectées à la tâche τ_1 et p_4 à la tâche τ_2. Donc t_2 et t_3 font partie de τ_1 et t_4 fait partie de τ_2. Par ailleurs, t_1 commence τ_1 et t_3 la finit ; t_2 commence τ_2 et t_4 la finit.

8.2.3 Fonction d'activité pour EDF

Bien que les changements de priorités correspondent à des changements de marquage, il est évident que pour un marquage donné, c'est-à-dire une configuration de tâche donnée, les priorités dépendent encore de l'avancement respectif des tâches, c'est-à-dire des valuations des transitions. Nous avons donc $Flow : P^{\mathbb{N}} \times \mathbb{R}^{|T|} \mapsto T^{\mathbb{R}^+}$. Soit $\mathcal{D} : Tasks \times P^{\mathbb{N}} \times \mathbb{R}^{|T|} \mapsto \mathbb{R}^+$ la fonction donnant pour une tâche et un état du réseau donnés, le temps restant jusqu'à l'échéance de la tâche. Alors, pour toute transition t du réseau, t est *active* si sa tâche associée $\gamma(t)$ est la plus proche de son échéance :

$$Flow(M,\nu)(t) = \begin{cases} 1 \text{ si } \mathcal{D}(\gamma(t),M,\nu) = \min_{\tau \in Tasks \text{ t.q. } \mathcal{P}(\tau)=\mathcal{P}(\gamma(t))}\{\mathcal{D}(\tau,M,\nu)\} \\ 0 \text{ sinon.} \end{cases}$$

Le calcul de la fonction d'activité repose donc sur la connaissance de la fonction $\mathcal{D}$.

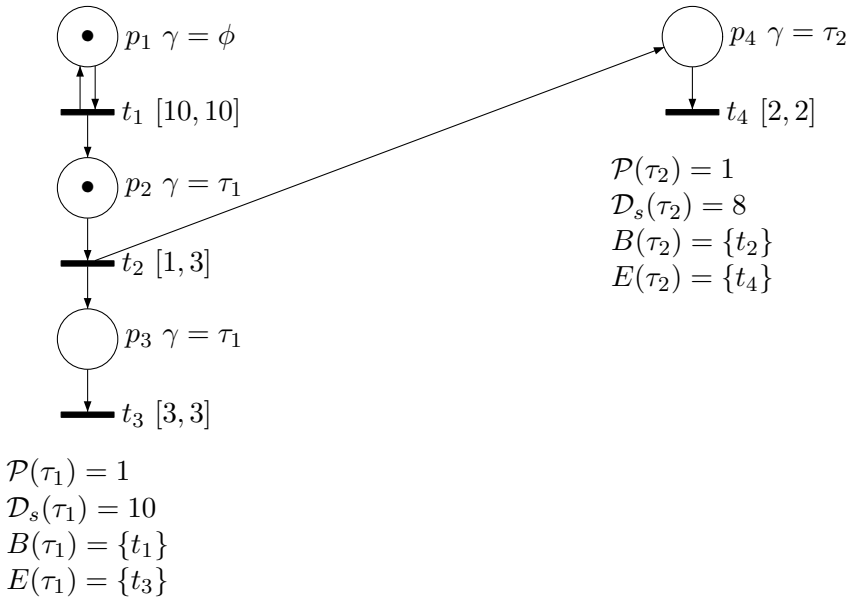

FIG. 8.8 – Un réseau de Petri T-temporel étendu à l'ordonnancement

Soit une tâche τ. Soit une séquence de tir $(M_1, \nu_1) \overset{d_1,t_1}{\rightarrow} \cdots \overset{d_{n-1},t_{n-1}}{\rightarrow} (M_n, \nu_n)$. Soit $j \in [\![1, n]\!]$ tel que $t_j \in B(\tau)$ et $\forall i \leq j, t_i \notin B(\tau)$. j est donc l'indice de de la première occurrence du tir d'une transition qui commence τ.

Soit k l'indice de l'occurrence du tir de la première transition, après t_j, qui finit τ alors,

$$\forall m \in [\![j, k+1]\!], \mathcal{D}(\tau, M_m, \nu_m) = \mathcal{D}_s(\tau) - \sum_{i \in [\![j+1,m]\!]} d_i \tag{8.1}$$

Par ailleurs, $\mathcal{D}(\tau, M_{k+1}, \nu_{k+1}) < 0$ implique que la tâche a manqué son échéance et donc que le système n'est pas ordonnançable.

8.2.4 Graphe des classes d'états

Nous allons maintenant calculer le graphe des classes d'états de ce modèle. Étant donné que la fonction *Flow* dépend des valuations des transitions, un nouveau problème survient en groupant les états en classes. Considérons à nouveau la figure 8.8. Si nous tirons t_2 à la date 1, nous obtenons un état s et $\mathcal{D}(\tau_1, s) = 9$ et $\mathcal{D}(\tau_2, s) = 8$ donc τ_2 est la tâche la plus prioritaire. Si, au contraire, nous tirons t_2 à la date 3, alors dans l'état s' obtenu, $\mathcal{D}(\tau_1, s') = 7$ et $\mathcal{D}(\tau_2, s') = 8$ donc c'est τ_1 qui est la tâche la plus prioritaire. Or, le calcul du graphe des classes d'états pour les *Scheduling*-TPN que nous venons de voir regroupe tous les états résultant du tir de t_2 dans une même classe. Par conséquent, nous n'avons plus forcément la même fonction d'activité pour tous les états d'une même classe. Nous allons donc redéfinir le partitionnement de l'espace d'états de façon à ce que l'application de la fonction *Flow* donne le même résultat pour tous les états d'une même partition.

Par ailleurs, comme nous l'avons déjà vu au chapitre 5 nous n'avons pas accès aux valuations des transitions dans les classes d'états, et donc nous ne pouvons pas calculer directement le temps restant pour chaque tâche jusqu'à son échéance.

Afin de pallier à ces deux problèmes, nous introduisons de nouvelles inéquations dans le domaine qui nous donnent ces temps.

Pour toute activation d'une tâche τ, nous ajoutons au domaine de tir la variable $\mathcal{D}_\tau$ qui représente une transition *fictive* du réseau que nous noterons également $\mathcal{D}_\tau$. Cette transition est *toujours active*. Nous pouvons donc considérer que c'est une transition observatrice de toute séquence de tir suivant le tir d'une transition commençant la tâche τ, au sens de la définition 41.

Définition 41 (Transition observatrice d'une séquence de tirs) *Soit* $\rho = s_1 = (M_1, \nu_1) \overset{t_1,d_1}{\to} s_2 \overset{t_2,d_2}{\to} \cdots \overset{t_{k-1},d_{k-1}}{\to} s_k$ *une séquence de tir d'un Scheduling-TPN* $\mathcal{T} = (P, T, {}^\bullet(), ()^\bullet, \alpha, \beta, M_0, Flow)$. *Une transition* $t \in T$ *est une* transition observatrice *de la séquence* ρ *ssi*

$$\begin{cases} \nu_1(t) = 0, \\ t \text{ reste active tout au long de } \rho, \\ \alpha(t) = \beta(t) = \Omega, \\ \text{avec } \Omega > \sum_{j \in [\![2,k-1]\!]} \beta(t_j). \end{cases}$$

En effet, soit $\rho = s_1 \overset{t_1,d_1}{\to} s_2 \overset{t_2,d_2}{\to} \cdots \overset{t_{k-1},d_{k-1}}{\to} s_k$ une séquence de tir telle que $t_1 \in B(\tau)$. Si $\mathcal{D}_\tau$ ne vérifie pas la définition 41 alors $\mathcal{D}_s(\tau) < \sum_{j \in [\![2,k-1]\!]} \beta(t_j)$ et donc le système n'est pas ordonnançable[1].

Par conséquent, nous pouvons appliquer le théorème 22, dont la preuve est donnée dans l'annexe A.

Théorème 22 (Valeur des observateurs) *Soit* $\rho = s_1 = (M_1, \nu_1) \overset{t_1,d_1}{\to} s_2 \overset{t_2,d_2}{\to} \cdots \overset{t_{k-1},d_{k-1}}{\to} s_k$ *une séquence de tir d'un Scheduling-TPN* $\mathcal{T} = (P, T, {}^\bullet(), ()^\bullet, \alpha, \beta, M_0, Flow)$. *Soit* $t_i \in T \setminus \{t_1, \ldots, t_{k-1}\}$ *une transition observatrice de* ρ. *Alors* $\Omega - \beta_i^{(k)} \leq d_1 + d_2 + \cdots + d_{k-1} \leq \Omega - \alpha_i^{(k)}$, *en notant* $[\alpha_i^{(k)}, \beta_i^{(k)}]$ *l'intervalle* $[\![D^{(k)}]\!]_i$ *des solutions projetées sur la variable* t_i *du domaine* $D^{(k)}$ *de la classe obtenue après la séquence de transitions* ρ *dans le graphe des classes d'états.*

De façon similaire, notons $[\alpha_{\mathcal{D}_\tau}^{(k)}, \beta_{\mathcal{D}_\tau}^{(k)}]$ l'intervalle $[\![D^{(k)}]\!]_{\mathcal{D}_\tau}$ des solutions projetées sur la variable $\mathcal{D}_\tau$ du domaine $D^{(k)}$ de la classe obtenue après la séquence de transitions ρ dans le graphe des classes d'états. Nous obtenons alors :

$$\mathcal{D}_s(\tau) - \beta_{\mathcal{D}_\tau}^{(k)} \leq d_2 + d_3 + \cdots + d_{k-1} \leq \mathcal{D}_s(\tau) - \alpha_{\mathcal{D}_\tau}^{(k)} \tag{8.2}$$

Ou encore :

$$\alpha_{\mathcal{D}_\tau}^{(k)} \leq \mathcal{D}_s(\tau) - (d_2 + d_3 + \cdots + d_{k-1}) \leq \beta_{\mathcal{D}_\tau}^{(k)} \tag{8.3}$$

Soit finalement :

$$\alpha_{\mathcal{D}_\tau}^{(k)} \leq \mathcal{D}(\tau, s_k) \leq \beta_{\mathcal{D}_\tau}^{(k)} \tag{8.4}$$

Donc la projection des solutions du domaine sur la variable $\mathcal{D}_\tau$ que nous avons ajoutée représente exactement l'ensemble des temps jusqu'à l'échéance de la tâche τ pour les états contenus dans la classe.

[1] Si le calcul d'une nouvelle classe donne des valeurs négatives de cette variable, nous pouvons en déduire que le système n'est pas ordonnançable et arrêter le calcul.

En ajoutant des contraintes du type $\mathcal{D}_\tau \geq \mathcal{D}_{\tau'}$ nous nous assurons donc que l'application de la fonction $Flow$ est cohérente[2] (donne le même résultat pour tous les états de la classe).

Classe initiale

En tenant compte de ces nouvelles variables, la classe initiale est définie par $(M_0, \{\alpha(t_i) \leq \theta_i \leq \beta_i, \forall i \in enabled(M_0)\} \cup \{\mathcal{D}_s(\tau) \leq \mathcal{D}_\tau \leq \mathcal{D}_s(\tau), \forall \tau \text{ t.q. } \exists t \in B(\tau) \text{ t.q. } t^\bullet < M_0\})$

Calcul des successeurs

L'algorithme de calcul des classes devient :

Soit une classe $C = (M, D)$ et une transition tirable t_f, la classe $C' = (M', D')$, successeur de C par t_f, est donnée par :

- Le nouveau marquage est calculé de façon classique par $M' = M - {}^\bullet t_f + t_f{}^\bullet$
- Le nouveau domaine de tir, D', est calculé selon les étapes suivantes. Nous le noterons $next(D, t_f)$
  1. changements de variables $\forall j,$ t.q. t_j est sensibilisée, $\theta_j = Flow(M)(t_j) * \theta_f + \theta'_j$;
  2. $\forall j \neq t_f$, ajout des contraintes $\theta'_j \geq 0$;
  3. élimination des variables correspondant à des transitions désensibilisées par le tir de t_f (ce qui inclut donc t_f), par exemple en utilisant la méthode de Fourier-Motzkin [Dan63] ainsi que des inéquations relatives à l'échéance des tâches qui se terminent ($\exists \tau \in Tasks$ t.q. $t \in E(\tau)$ et t est désensibilisée par le tir de t_f).
  4. ajout des inéquations relatives aux transitions nouvellement sensibilisées par le tir de t_f, ainsi que des inéquations relatives aux échéances des tâches qui commencent :

$$\forall t_k \in\uparrow enabled(M, t_f), \begin{cases} \alpha(t_k) \leq \theta'_k \leq \beta(t_k), \\ \text{et si } \exists \tau \in Tasks \text{ t.q. } t_f \in B(\tau), \mathcal{D}_s(\tau) \leq \mathcal{D}_\tau \leq \mathcal{D}_s(\tau). \end{cases}$$

  5. détermination de la forme canonique D'^* du nouveau domaine de tir.
  6. pour toute tâche τ qui commence, c'est-à-dire telle que $t_f \in B(\tau)$, si $\exists \tau'$ t.q. $\mathcal{D}_s(\tau) \in [\![D'^*]\!]_{\mathcal{D}_{\tau'}}$, alors nous dupliquons la classe, une instance recevant l'inéquation supplémentaire $\mathcal{D}_\tau \leq \mathcal{D}_{\tau'}$ et l'autre $\mathcal{D}_\tau \geq \mathcal{D}_{\tau'}$.

8.2.5 Exemple

Nous allons calculer le graphe des classes de l'exemple de la figure 8.8. Dans tout cet exemple, les inéquations redondantes ne seront pas explicitées.

La classe initiale est

$$C_0 = \begin{cases} \{p_1, p_2\} \\ 10 \leq \theta_1 \leq 10 \\ 1 \leq \theta_2 \leq 3 \\ 10 \leq \mathcal{D}_{\tau_1} \leq 10 \end{cases}$$

[2] En toute rigueur, le cas $\mathcal{D}_\tau = \mathcal{D}_{\tau'}$ est problématique car les deux priorités sont alors égales. Nous considérerons que le choix de la tâche à exécuter est fait de façon arbitraire puisque, de toute façon, les deux possibilités sont explorées. Nous pourrions aussi ajouter des inéquations strictes et distinguer le cas d'égalité en lui appliquant *Round-Robin*.

Nous avons $Flow(t_1) = Flow(t_2) = 1$. La seule transition tirable est t_2. Nous la tirons et nous obtenons :

$$C_1 = \begin{cases} \{p_1, p_3, p_4\} \\ 7 \leq \theta_1 \leq 9 \\ 3 \leq \theta_3 \leq 3 \\ 2 \leq \theta_4 \leq 2 \\ 7 \leq \mathcal{D}_{\tau_1} \leq 9 \\ 8 \leq \mathcal{D}_{\tau_2} \leq 8 \\ 0 \leq \theta_1 - \mathcal{D}_{\tau_1} \leq 0 \end{cases}$$

t_2 a été tirée et c'est une transition qui commence τ_2 donc nous avons ajouté les inéquations $\mathcal{D}_s(\tau_2) \leq \mathcal{D}_{\tau_2} \leq \mathcal{D}_s(\tau_2)$. De plus $\mathcal{D}_s(\tau_2) \in [\![D_1]\!]_{\mathcal{D}_{\tau_1}}$ donc nous devons partitionner la classe C_1 en C_1' et C_1'' :

$$C_1' = \begin{cases} \{p_1, p_3, p_4\} \\ 7 \leq \theta_1 \leq 9 \\ 3 \leq \theta_3 \leq 3 \\ 2 \leq \theta_4 \leq 2 \\ 7 \leq \mathcal{D}_{\tau_1} \leq 9 \\ 8 \leq \mathcal{D}_{\tau_2} \leq 8 \\ 0 \leq \theta_1 - \mathcal{D}_{\tau_1} \leq 0 \\ \mathcal{D}_{\tau_1} \leq \mathcal{D}_{\tau_2} \end{cases} \qquad C_1'' = \begin{cases} \{p_1, p_3, p_4\} \\ 7 \leq \theta_1 \leq 9 \\ 3 \leq \theta_3 \leq 3 \\ 2 \leq \theta_4 \leq 2 \\ 7 \leq \mathcal{D}_{\tau_1} \leq 9 \\ 8 \leq \mathcal{D}_{\tau_2} \leq 8 \\ 0 \leq \theta_1 - \mathcal{D}_{\tau_1} \leq 0 \\ \mathcal{D}_{\tau_2} \leq \mathcal{D}_{\tau_1} \end{cases}$$

Dans la classe C_1', nous avons $Flow(t_3) = 1$ et $Flow(t_4) = 0$. Seule t_3 est tirable. Nous tirons donc t_3 et obtenons :

$$C_2 = \begin{cases} \{p_1, p_4\} \\ 4 \leq \theta_1 \leq 6 \\ 2 \leq \theta_4 \leq 2 \\ 5 \leq \mathcal{D}_{\tau_2} \leq 5 \end{cases}$$

t_3 étant une transition qui finit τ_1, son tir nous permet de retirer les inéquations relatives à $\mathcal{D}_{\tau_1}$. Dans C_2, $Flow(t_4) = 1$ et t_4 est la seule transition tirable. Les tirs successifs de t_4 et t_1 conduisent sans difficulté à la classe initiale C_0.

Dans la classe C_1'', nous avons $Flow(t_3) = 0$ et $Flow(t_4) = 1$. Seule t_4 est tirable. Nous tirons donc t_4 et obtenons :

$$C_3 = \begin{cases} \{p_1, p_3\} \\ 5 \leq \theta_1 \leq 7 \\ 3 \leq \theta_3 \leq 3 \\ 5 \leq \mathcal{D}_{\tau_1} \leq 7 \\ 0 \leq \theta_1 - \mathcal{D}_{\tau_1} \leq 0 \end{cases}$$

Comme précédemment, t_4 finit τ_2 donc son tir entraîne le retrait de $\mathcal{D}_{\tau_2}$ du domaine de C_3. Puis, la séquence t_3, t_1 conduit à la classe initiale.

Finalement nous obtenons le graphe (non déterministe) de la figure 8.9.

8.2.6 Bilan

Dans cette section, nous avons présenté une modèle pour des systèmes ordonnancés par la politique *Earliest Deadline first*. Nous avons également adapté le calcul du graphe des

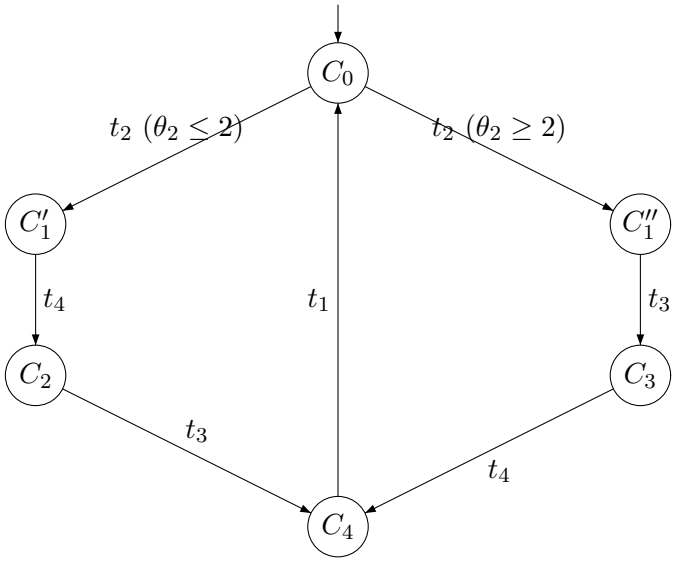

FIG. 8.9 – Graphe des classes du réseau de la figure 8.8.

classes proposé au chapitre 6 à ce modèle. Cela implique notamment un partionnement supplémentaire des classes d'états.

Comme pour le modèle du chapitre 6, l'accessibilité, la bornitude et les autres propriétés « intéressantes » sont indécidables et le calcul de l'espace d'états proposé est donc un semi-algorithme.

8.3 Conclusion

Dans ce chapitre, nous avons vu comment étendre le modèle des réseaux de Petri à l'ordonnancement pour prendre en compte deux politiques classiques d'ordonnancement : *Round-Robin* et *Earliest Deadline First*. Pour ces deux modèles, le calcul du graphe des classes s'étend naturellement et permet de calculer l'espace d'états.

Il pourra être intéressant d'étudier l'adaptation de la méthode du chapitre 7 aux cas où le *Round-Robin* est utilisé afin d'obtenir un automate hybride linéaire temporellement bisimilaire au réseau initial. Dans le cas de *Earliest Deadline First*, le calcul de l'automate à stopwatch du chapitre 7 est *a priori* valable en utilisant le partionnement supplémentaire proposé dans ce chapitre.

Par ailleurs, les modèles sont "compatibles" d'un point de vue théorique, un travail futur pourra consister à réaliser une implémentation permettant de prendre en compte chacune des politiques proposées. On pourra alors vérifier, par exemple, une application de trois processeurs : les deux premiers avec une politique d'ordonnancement à priorités fixes (l'un avec *Round-Robin*, l'autre sans) et un troisième processeur avec une politique d'ordonnancement *Earliest Deadline First*.

Quatrième partie

Application

Chapitre 9

Implémentation

Dans ce chapitre, nous présentons brièvement l'outil de vérification ROMEO. Puis, nous nous intéressons plus particulièrement à deux aspects spécifiques de l'implémentation.

9.1 Roméo

ROMEO est un outil de vérification dédié aux réseaux de Petri T-temporels. Il est disponible librement à l'adresse suivante :
http ://www.irccyn.ec-nantes.fr/irccyn/d/en/equipes/TempsReel/logs/software-2-romeo

Il se compose d'une interface graphique programmée en Tcl/Tk et de deux modules de calcul, GPN2 et MERCUTIO programmés en C++. MERCUTIO permet la traduction des réseaux de Petri T-temporels bornés en automates temporisés par la méthode basée sur un graphe des zones adapté de celui effectué sur les automates temporisés [GRR05]. L'interface graphique propose également une implémentation de la traduction structurelle de [CR04].

J'ai implémenté dans GPN2 la plupart des semi-algorithmes décrits dans cette thèse. Plus précisément, les algorithmes proposés sont :

- Pour les réseaux de Petri autonomes, le graphe des marquages ;
- Pour les réseaux de Petri T-temporels :
  - Le graphe des classes d'états (chapitre 4) ;
  - L'automate temporisé des classes d'états (chapitre 5) ;
- Pour les réseaux de Petri T-temporels étendus à l'ordonnancement :
  - Le graphe des classes d'états (chapitre 6), de façon exacte[1] et avec la surapproximation à base de DBM ;
  - L'automate à *stopwatches* des classes d'états (chapitre 7) avec la surapproximation à base de DBM.

Les réseaux de Petri fournis en entrée du programme sont définis en XML. Les résultats sont fournis aux formats ALDEBARAN [Fer89] et MEC-V [Vin03] pour les différents graphes, UPPAAL et KRONOS pour l'automate temporisé des classes d'états et HYTECH pour l'automate à *stopwatches* des classes d'états.

Enfin, dans le cadre des réseaux de Petri T-temporels, nous proposons deux améliorations du calcul et de la représentation des classes d'états.

[1] Cette implémentation a été effectuée par Morgan MAGNIN au cours de son DEA

9.2 Améliorations du calcul des successeurs

9.2.1 Calcul en $O(n^2)$ des classes d'états

Dans cette sous-section, nous allons présenter une amélioration de la complexité du calcul des successeurs d'une classe d'états que nous avons développée. Un résultat tout à fait similaire a été obtenu par Hanifa Boucheneb et John Mullins, et publié dans [BM03].

Nous avons vu dans la sous-section 4.3.2 l'algorithme de calcul des classes d'états. L'opération la plus coûteuse est la mise sous forme canonique, habituellement faite par l'algorithme de Floyd-Warshall [CLR90] et de complexité temporelle $O(n^3)$ où n est la taille du domaine de tir, c'est-à-dire le nombre de transitions sensibilisées par le marquage de la classe. En conséquence, le calcul classique d'une classe d'états est de complexité temporelle $O(n^3)$.

Nous proposons un calcul simplifié de complexité $O(n^2)$. Cela est possible grâce à la forme particulière des DBM représentant les domaines de tir des classes et au fait que chaque nouveau domaine est calculé à partir d'un domaine sous forme canonique.

Algorithme de Floyd-Warshall L'algorithme de Floyd-Warshall consiste à calculer les plus courts chemins dans un graphe de contraintes [Ber01]. Cet algorithme peut être écrit pour les classes d'états des réseaux de Petri T-temporels sous la forme suivante :

for k=1 to N **do**
  for i=1 to N **do**
    for j=1 to N **do**
      $\alpha_j \leftarrow \max\{\alpha_j, \alpha_k - \gamma_{kj}\}$
      $\beta_i \leftarrow \min\{\beta_i, \beta_k + \gamma_{ik}\}$
      $\gamma_{ij} \leftarrow \min\{\gamma_{ij}, \gamma_{ik} + \gamma_{kj}\}$
      $\gamma_{ij} \leftarrow \min\{\gamma_{ij}, \beta_i - \alpha_j\}$
    end for
  end for
end for

En conséquence, soit $C = (M, D)$ une classe d'états telle que

$$D = \begin{cases} \forall i \in [1, N], \alpha_i \leq \theta_i \leq \beta_i, \\ \forall i, j \in [1, N], -\gamma_{ji} \leq \theta_i - \theta_j \leq \gamma_{ij} \end{cases} \tag{9.1}$$

Alors si D est sous forme canonique, les équations (9.2) sont vérifiées :

$$\begin{cases} \forall i, \forall k, \alpha_i \geq \alpha_k - \gamma_{ki}, \\ \forall i, \forall k, \beta_i \leq \beta_k + \gamma_{ik}, \\ \forall i, j, \forall k, \gamma_{ij} \leq \gamma_{ik} + \gamma_{kj}, \\ \forall i, j, \forall k, \gamma_{ij} \leq \beta_i - \alpha_j \end{cases} \tag{9.2}$$

Forme intermédiaire Nous allons maintenant démontrer le lemme suivant :

Lemme 8 (Forme intermédiaire) *Soit $C = (M, D)$ une classe d'états telle que D, sous forme canonique, est*

$$\begin{cases} \forall i \in [1, N], \alpha_i \leq \theta_i \leq \beta_i, \\ \forall i, j \in [1, N], -\gamma_{ji} \leq \theta_i - \theta_j \leq \gamma_{ij} \end{cases} \tag{9.3}$$

Alors le domaine D', pas encore sous forme canonique, de la classe C' obtenue par le tir de la transition t_f depuis C est :

$$\begin{cases} \forall i, \alpha'_i = \max\{0, -\gamma_{fi}\}, \\ \forall i, \beta'_i = \gamma_{if}, \\ \forall i, j, \gamma'_{ij} = \gamma_{ij} \end{cases} \tag{9.4}$$

Preuve. La preuve qui suit serait la même avec N variables ($N \geq 3$), aussi dans un souci de simplicité, nous l'écrirons avec seulement trois, ce qui est le minimum pour obtenir tous les types contraintes.

Soit $C = (M, D)$ une classe d'états telle que D, sous forme canonique, est :

$$\begin{cases} \alpha_1 \leq \theta_1 \leq \beta_1, \\ \alpha_2 \leq \theta_2 \leq \beta_2, \\ \alpha_3 \leq \theta_3 \leq \beta_3, \\ -\gamma_{21} \leq \theta_1 - \theta_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq \theta_1 - \theta_3 \leq \gamma_{13}, \\ -\gamma_{32} \leq \theta_2 - \theta_3 \leq \gamma_{23} \end{cases} \tag{9.5}$$

On calcule alors la classe $C' = (M', D')$ obtenue par le tir de la la transition t_1 depuis C. La première étape du calcul de C' est les changements de variables $\theta_2 = \theta'_2 + \theta_1$ et $\theta_3 = \theta'_3 + \theta_1$. On écrit également le système en fonction de θ_1 afin d'appliquer la méthode d'élimination de variables de Fourier-Motzkin [Dan63] à θ_1. Enfin, on ajoute les contraintes $\theta'_2 \geq 0$ et $\theta'_3 \geq 0$, les instants de tirs des transitions ne pouvant être négatifs, sinon cela signifie que la transition correspondante aurait dû être tirée avant t_1.

$$\begin{cases} \theta_1 \geq \alpha_1, \\ \theta_1 \geq \alpha_2 - \theta'_2, \\ \theta_1 \geq \alpha_3 - \theta'_3, \\ \theta_1 \leq \beta_1, \\ \theta_1 \leq \beta_2 - \theta'_2, \\ \theta_1 \leq \beta_3, -\theta'_3, \\ -\gamma_{21} \leq -\theta'_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq -\theta'_3 \leq \gamma_{13}, \\ -\gamma_{32} \leq \theta'_2 - \theta'_3 \leq \gamma_{23}, \\ \theta'_2 \geq 0, \\ \theta'_3 \geq 0 \end{cases} \tag{9.6}$$

La méthode de Fourier-Motzkin consiste à écrire que ce système est équivalent à celui obtenu en écrivant que tous les minorants de θ_1 sont inférieurs ou égaux à tous ses majorants. Après quelques simplifications, nous obtenons :

$$\begin{cases} \max\{0, -\gamma_{12}, \alpha_2 - \beta_1\} \leq \theta'_2 \leq \min\{\gamma_{21}, \beta_2 - \alpha_1\}, \\ \max\{0, -\gamma_{13}, \alpha_3 - \beta_1\} \leq \theta'_3 \leq \min\{\gamma_{31}, \beta_3 - \alpha_1\}, \\ \max\{-\gamma_{32}, \alpha_2 - \beta_3\} \leq \theta'_2 - \theta'_3 \leq \min\{\gamma_{23}, \beta_2 - \alpha_3\} \end{cases} \tag{9.7}$$

Puisque D est sous forme canonique, il vérifie les équations (9.2), d'où finalement :

$$\begin{cases} \max\{0, -\gamma_{12}\} \leq \theta'_2 \leq \gamma_{21}, \\ \max\{0, -\gamma_{13}\} \leq \theta'_3 \leq \gamma_{31}, \\ -\gamma_{32} \leq \theta'_2 - \theta'_3 \leq \gamma_{23} \end{cases} \tag{9.8}$$

□

Calcul de la classe suivante Enfin, on donne une expression directe du domaine d'une classe sous forme canonique en fonction du domaine sous forme canonique de son prédécesseur. Cette expression se calcule en $O(N^2)$.

Théorème 23 (Classe suivante) *Soit $C = (M, D)$ une classe d'états telle que D, sous sa forme canonique, s'écrit :*

$$\begin{cases} \forall i \in [1, N], \alpha_i \leq \theta_i \leq \beta_i, \\ \forall i, j \in [1, N], -\gamma_{ji} \leq \theta_i - \theta_j \leq \gamma_{ij} \end{cases} \tag{9.9}$$

Alors le domaine D', sous sa forme canonique, de la classe C' obtenue par le tir de la transition t_f depuis C est :

$$\begin{cases} Z' = \{i \in [1, N], \alpha'_i = 0\}, \\ \forall i, \alpha'_i = \max\{0, -\gamma_{fi}, \max_{z \in Z'}\{-\gamma_{zi}\}\}, \\ \forall i, \beta'_i = \gamma_{if}, \\ \forall i, j, \gamma'_{ij} = \min\{\gamma_{ij}, \beta'_i - \alpha'_j\} \end{cases} \tag{9.10}$$

Preuve. L'élimination des variables correspondant à des transitions désensibilisées peut se faire après le calcul de la forme canonique de la nouvelle classe. Il suffit alors de supprimer toutes les inéquations relatives à ces variables. De même l'ajout des variables correspondant à des transitions nouvellement sensibilisées peut également se faire après le calcul de la forme canonique. Le système ainsi obtenu reste sous forme canonique [Ber01].

Nous pouvons donc partir de la forme intermédiaire de l'équation (9.4) :

$$\begin{cases} \forall i, \alpha'_i = \max\{0, -\gamma_{fi}\}, \\ \forall i, \beta'_i = \gamma_{if}, \\ \forall i, j, \gamma'_{ij} = \gamma_{ij} \end{cases} \tag{9.11}$$

Nous allons procéder par récurrence sur la boucle externe de l'algorithme de Floyd-Warshall. Nous allons montrer que $\forall K \in [\![1, N]\!]$.

$$D'^K = \begin{cases} Z'^K = \{i \in [1, K], \alpha'_i = 0\}, \\ \forall i, \alpha'^K_i = \max\{0, -\gamma_{fi}, \max_{z \in Z'^K}\{-\gamma_{zi}\}\}, \\ \forall i, \beta'^K_i = \gamma_{if}, \\ \forall i, j, \gamma'^K_{ij} = \min\{\gamma_{ij}, \beta'^K_i - \alpha'^K_j\} \end{cases} \tag{9.12}$$

où l'exposant K sur une variable dénote sa valeur à la fin de l'itération K de l'algorithme. L'exposant 0 désigne donc logiquement la valeur d'une variable avant l'exécution de l'algorithme, c'est-à-dire la valeur donnée par la forme intermédiaire.

Montrons d'abord que cette expression est vraie à la fin de la première itération, c'est-à-dire pour $K = 1$. Soit $(i, j) \in [\![1, N]\!]^2$.

- instruction $\alpha'^1_i \leftarrow \max\{\alpha'^0_i, \alpha'^0_1 - \gamma'^0_{1i}\}$. Nous savons que $\gamma'^0_{1i} = \gamma_{1i}$
  - si $\alpha'^0_1 = -\gamma_{f1}$ alors avec (9.2), $\alpha'^0_1 - \gamma'^0_{1i} \leq -\gamma_{fi} \leq \alpha'^0_i$.
  - si $\alpha'^0_1 = 0$ alors $\alpha'^0_1 - \gamma'^0_{1i} = -\gamma_{1i}$. Or $-\gamma_{1i}$ peut tout à fait être plus grand que α'^0_i, donc dans ce cas l'instruction peut être utile.

En conclusion, l'instruction $\alpha_i'^1 \leftarrow \max\{\alpha_i'^0, \alpha_1'^0 - \gamma_{1i}'^0\}$ ne peut modifier la valeur de $\alpha_i'^1$ que si $\alpha_1'^0 = 0$. Et donc $\alpha_i'^1 = \max\{0, -\gamma_{fi}, \max_{z\in Z'^1}\{-\gamma_{zi}\}\}$.

- instruction $\beta_i'^1 \leftarrow \min\{\beta_i'^0, \beta_1'^0 + \gamma_{i1}'^0\}$. Nous savons que $\gamma_{i1}'^0 = \gamma_{i1}$ alors $\beta_1'^0 + \gamma_{i1}'^0 = \gamma_{1f} + \gamma_{i1}$ donc avec (9.2), $\beta_1'^0 + \gamma_{i1}'^0 \geq \gamma_{if} \geq \beta_i'^0$. Donc cette instruction ne peut pas modifier la valeur de $\beta_i'^1$. Donc $\beta_i'^1 = \gamma_{if}$.
- instruction $\gamma_{ij}'^1 \leftarrow \min\{\gamma_{ij}'^0, \gamma_{i1}'^0 + \gamma_{1j}'^0\}$. Nous savons que $\gamma_{i1}'^0 = \gamma_{i1}$ et $\gamma_{1j}'^0 = \gamma_{1j}$ alors $\gamma_{i1}'^0 + \gamma_{1j}'^0 = \gamma_{i1} + \gamma_{1j}$ et donc avec (9.2), $\gamma_{i1}'^0 + \gamma_{1j}'^0 \geq \gamma_{ij} \geq \gamma_{ij}'^0$. Donc cette instruction est inutile.
- instruction $\gamma_{ij}'^1 \leftarrow \min\{\gamma_{ij}'^0, \beta_i'^1 - \alpha_j'^1\}$.
  - si $\alpha_j'^1 = -\gamma_{fj}$ alors $\beta_i'^1 - \alpha_j'^1 = \gamma_{if} + \gamma_{fj}$ donc $\beta_i'^1 - \alpha_j'^1 \geq \gamma_{ij} \geq \gamma_{ij}'^0$.
  - si $\alpha_j'^1 = 0$ alors $\beta_i'^1 - \alpha_j'^1 = \gamma_{if}$ Il est possible que γ_{if} soit inférieur à $\gamma_{ij}'^1$.

  En conclusion, il est possible que $\beta_i'^1 - \alpha_j'^1$ soit inférieur à $\gamma_{ij}'^1$ donc $\gamma_{ij}'^1 = \min\{\gamma_{ij}, \beta_i'^1 - \alpha_j'^1\}$.

Nous avons montré que la propriété est vraie pour $K = 1$. Supposons maintenant que la propriété est vraie pour $K \in [\![0, N]\!]$. Nous allons montrer que cette expression est vérifiée pour l'itération $K + 1$. Soit $(i, j) \in [\![1, N]\!]^2$.

- instruction $\alpha_i'^K \leftarrow \max\{\alpha_i'^K, \alpha_K'^K - \gamma_{Ki}'^K\}$.
  - supposons que $\gamma_{Ki}'^K = \gamma_{Ki}$
    - si $\alpha_K'^K = -\gamma_{fK}$ alors avec (9.2), $\alpha_K'^K - \gamma_{Ki}'^K \leq -\gamma_{fi} \leq \alpha_i'^K$.
    - si $\exists z \in Z'^K, \alpha_K'^K = -\gamma_{zK}$ alors avec (9.2), $\alpha_K'^K - \gamma_{Ki}'^K \leq -\gamma_{zi} \leq \alpha_i'^K$.
    - si $\alpha_K'^K = 0$ alors $\alpha_K'^K - \gamma_{Ki}'^K = -\gamma_{Ki}$. Or $-\gamma_{Ki}$ peut tout à fait être plus grand que $\alpha_i'^K$, donc dans ce cas l'instruction peut être utile.
  - supposons que $\gamma_{Ki}'^K = \beta_K'^K - \alpha_i'^K$ alors $\alpha_K'^K - \gamma_{Ki}'^K = \alpha_i'^K - (\beta_K'^K - \alpha_K'^K)$. Or $\beta_K'^K - \alpha_K'^K \leq 0$, donc $\alpha_K'^K - \gamma_{Ki}'^K \leq \alpha_i'^K$.

  En conclusion, l'instruction $\alpha_i'^K \leftarrow \max\{\alpha_i'^K, \alpha_K'^K - \gamma_{Ki}'^K\}$ ne peut modifier la valeur de $\alpha_i'^K$ que si $\alpha_K'^K = 0$. Et donc $\alpha_i'^{K+1} = \max\{0, -\gamma_{fi}, \max_{z\in Z'^{K+1}}\{-\gamma_{zi}\}\}$.
- instruction $\beta_i'^K \leftarrow \min\{\beta_i'^K, \beta_K'^K + \gamma_{iK}'^K\}$.
  - si $\gamma_{iK}'^K = \gamma_{iK}$ alors $\beta_K'^K + \gamma_{iK}'^K = \gamma_{Kf} + \gamma_{iK}$ donc avec (9.2), $\beta_K'^K + \gamma_{iK}'^K \geq \gamma_{if} \geq \beta_i'^K$.
  - si $\gamma_{iK}'^K = \beta_i'^K - \alpha_j'^K$ alors $\beta_K'^K + \gamma_{iK}'^K = (\beta_K'^K - \alpha_K'^K) + \beta_i'^K$. Comme $\beta_K'^K - \alpha_K'^K \geq 0$, $\beta_K'^K + \gamma_{iK}'^K \geq \beta_i'^K$.

  En conclusion, cette instruction ne peut pas modifier la valeur de $\beta_i'^K$. Donc $\beta_i'^{K+1} = \gamma_{if}$.
- instruction $\gamma_{ij}'^K \leftarrow \min\{\gamma_{ij}'^K, \gamma_{iK}'^K + \gamma_{Kj}'^K\}$.
  - supposons que $\gamma_{iK}'^K = \gamma_{iK}$
    - si $\gamma_{Kj}'^K = \gamma_{Kj}$ alors $\gamma_{iK}'^K + \gamma_{Kj}'^K = \gamma_{iK} + \gamma_{Kj}$ et donc avec (9.2), $\gamma_{iK}'^K + \gamma_{Kj}'^K \geq \gamma_{ij} \geq \gamma_{ij}'^K$.
    - si $\gamma_{Kj}'^K = \beta_K'^K - \alpha_j'^K$ alors $\gamma_{iK}'^K + \gamma_{Kj}'^K = \gamma_{iK}'^K + \beta_K'^K - \alpha_j'^K$. Or nous avons vu que $\gamma_{iK}'^K + \beta_K'^K \geq \beta_i'^K$ donc $\gamma_{iK}'^K + \gamma_{Kj}'^K \geq \beta_i'^K - \alpha_j'^K$. Nous sommes dans le cas de l'instruction suivante.
  - supposons que $\gamma_{iK}'^K = \beta_i'^K - \alpha_K'^K$
    - si $\gamma_{Kj}'^K = \gamma_{Kj}$ alors c'est un cas symétrique au précédent : nous sommes dans le cas de l'instruction suivante.
    - si $\gamma_{Kj}'^K = \beta_K'^K - \alpha_j'^K$ alors $\gamma_{iK}'^K + \gamma_{Kj}'^K = \beta_i'^K - \alpha_j'^K + (\beta_K'^K - \alpha_K'^K)$. Comme $\beta_K'^K - \alpha_K'^K \geq 0$, nous avons à nouveau $\gamma_{iK}'^K + \gamma_{Kj}'^K \geq \beta_i'^K - \alpha_j'^K$. Nous sommes dans le cas de l'instruction suivante.

  En conclusion, cette instruction est inutile soit directement soit par redondance avec la suivante.

– instruction $\gamma'^K_{ij} \leftarrow \min\{\gamma'^K_{ij}, \beta'^K_i - \alpha'^K_j\}$.
  – si $\alpha'^K_j = -\gamma_{fj}$ alors $\beta'^K_i - \alpha'^K_j = \gamma_{if} + \gamma_{fj}$ donc $\beta'^K_i - \alpha'^K_j \geq \gamma_{ij} \geq \gamma'^K_{ij}$.
  – si $\exists z \in Z'^K, \alpha'^K_j = -\gamma_{zj}$ alors $\beta'^K_i - \alpha'^K_j = \gamma_{if} + \gamma_{zj}$. Il est possible que $\gamma_{if} + \gamma_{zj}$ soit inférieur à γ'^K_{ij}.
  – si $\alpha'^K_j = 0$ alors $\beta'^K_i - \alpha'^K_j = \gamma_{if}$ Il est possible que γ_{if} soit inférieur à γ'^K_{ij}.

  En conclusion, il est possible que $\beta'^K_i - \alpha'^K_j$ soit inférieur à γ'^K_{ij} donc $\gamma'^{K+1}_{ij} = \min\{\gamma_{ij}, \beta'^K_i - \alpha'^K_j\}$.

Nous avons donc vérifié l'hypothèse de récurrence. Puisque la propriété est également vraie pour la forme intermédiaire à laquelle nous appliquons l'algorithme, nous avons prouvé le théorème 23. □

9.2.2 Limiter l'occupation mémoire des classes d'états

Pour de « gros » systèmes, il n'est pas rare que le premier facteur limitant ne soit pas le temps de calcul mais plutôt l'occupation en mémoire des résultats. Dans ce cas précis, par « gros », nous entendons « pour lesquels le nombre de transitions sensibilisées simultanément est grand ». En effet l'occupation mémoire d'une classe d'états est de l'ordre de $N + (n+1)^2$ où N est le nombre de places du réseaux et correspond au vecteur de marquage et n est le nombre de transitions sensibilisées par ce marquage. $(n+1)^2$ est donc classiquement la taille de la DBM représentant le domaine de tir.

Nous proposons donc une méthode pour limiter l'occupation mémoire d'une classe d'états en ne représentant pas explicitement les différences entre les dates de tir des transitions. En contrepartie, le calcul sera un peu plus lent.

Nous avons vu dans la sous-section précédente qu'une classe peut être calculée à partir de son prédécesseur par l'expression :

$$\begin{cases} Z' = \{i \in [1,N], \alpha'_i = 0\}, \\ \forall i, \alpha'_i = \max\{0, -\gamma_{fi}, \max_{z \in Z'}\{-\gamma_{zi}\}\}, \\ \forall i, \beta'_i = \gamma_{if}, \\ \forall i,j, \gamma'_{ij} = \min\{\gamma_{ij}, \beta'_i - \alpha'_j\} \end{cases} \tag{9.13}$$

Une conséquence de cette expression est que quelque soit le coefficient de la matrice γ, celui-ci peut-être exprimé en fonction des vecteurs α et β du domaine de tir de l'un de ses prédécesseur.

Autrement dit, soit une séquence de tir de transitions du réseau menant de l'état initial s_0 à l'état s_{n+1} par le tir des transitions $t_0, \ldots, t_n$. Notons la $s_0 \xrightarrow{t_0} \cdots \xrightarrow{t_n} s_{n+1}$. Il lui correspond un chemin dans le graphe des classes d'états que nous notons $C_0 \xrightarrow{t_0} \cdots \xrightarrow{t_n} C_{n+1}$. Pour une classe $C_k = (M_k, D_k)$, notons α^k_i, β^k_i et γ^k_{ij} les coefficients des inéquations du domaine de tir D_k. Alors :

$$\forall i,j, \exists p_{ij} \in [\![0, n+1]\!] \text{ tel que } \gamma^{n+1}_{ij} = \beta^{p_{ij}}_i - \alpha^{p_{ij}}_j \tag{9.14}$$

Par conséquent, afin de ne pas mémoriser explicitement la matrice γ, pour chaque coefficient, nous pourrions mémoriser l'indice de la classe dans laquelle il a été dernièrement modifié par rapport à la classe précédente. Bien sûr, ce n'est pas avantageux puisque cela représente au moins la même quantité d'information que γ.

Cependant, nous pouvons remarquer, dans la preuve du théorème 23, que lorsque l'un coefficient γ_{ij} est modifié, cela est uniquement dû au fait que α_j est modifié par rapport à

la forme intermédiaire (9.4) ou est nul. Il suffit donc de garder, pour chaque transition t_j, l'indice des classes dans lesquelles α_j a été modifié ou est nul, ainsi que l'indice des classes dans lesquelles t_i et t_j ont été sensibilisées. Cela nous donne pour tout j une liste d'indices L_j. Pour tout i, on a alors $\gamma_{ij} = \min_{p \in L_j}\{\beta_i^p - \alpha_j^p\}$.

Les longueurs des listes L_j ne sont pas forcément inférieures au nombre de transitions sensibilisées. Nous n'avons donc pas forcément un gain au niveau de l'occupation mémoire. Pour améliorer cela, nous pouvons remarquer que l'intersection des listes L_j a de bonnes chances d'être non vide. Par conséquent, nous allons fusionner les L_j de façon à obtenir une liste ordonnée. Soit $L = \bigcup_j L_j$, il suffit alors de mémoriser, pour chaque transition t_j, uniquement l'indice minimum p_j de la liste L_j correspondante. Il s'agit en fait de l'indice de la classe où la transition t_j a été sensibilisée. Nous avons donc finalement, pour tout i, j, $\gamma_{ij} = \min_{p \in L, p \geq p_j}\{\beta_i^p - \alpha_j^p\}$.

L'occupation mémoire du domaine d'une classe d'états, en utilisant cette représentation, est donc de l'ordre de $(3n + |L|)$, où $|L|$ est la longueur de L. Cette longueur correspond au nombre de classes générées, dans lesquelles l'un au moins des γ_{ij} a été modifié, depuis que la plus « vieille » des transitions sensibilisées a été sensibilisée. Nous avons vu au début de cette sous-section que pour une DBM, l'occupation mémoire est de l'ordre de $(n^2 + 2n)$. Dans le cas général, nous ne pouvons pas affirmer que $|L| + n < n^2$. Cependant, le nombre maximum de classes pendant lequel toute transition reste sensibilisée est un majorant de L. Et, en pratique, ce nombre est « petit ».

Cette méthode a été implémentée dans Romeo. Cependant, l'implémentation est encore trop expérimentale pour envisager la réalisation de tests pertinents.

En conclusion, nous avons présenté une représentation du domaine des classes d'états pour tenter de limiter l'espace mémoire qu'elles occupent. Contre un surcoût en temps de calcul, nous pouvons espérer une occupation mémoire pseudo linéaire en le nombre de transitions sensibilisées contre quadratique pour le codage classique sous forme d'une DBM.

9.3 Conclusion

Dans cette partie nous avons brièvement présenté l'outil implémentant les semi-algorithmes de calcul d'espaces d'états développés dans cette thèse. Nous avons également proposé une amélioration du calcul des classes d'états le rendant plus efficace et, à partir de cette amélioration, une représentation alternative du domaine des classes d'états pour essayer de limiter leur occupation mémoire.

Les perspectives de développement de Romeo incluent, à court terme, l'utilisation du format standard PNML [BCvH+03] et la fusion de Gpn2 et Mercutio.

Chapitre 10

Étude de cas

Depuis la norme `ISO 11783` *Agriculture and Forestry* basée sur `SAE J1939` (*CAN Format Version 2.0B*) certains constructeurs et équipementiers de tracteurs (véhicules agricoles) commencent à utiliser le CAN-bus (2.0B) et le *Agricultural Bus System* (LBS). C'est le cas de FENDT pour lequel les *Unité de Contrôle Électronique* (ECU) sont à base de processeurs INFINEON c167.

Dans ce chapitre, nous présentons une modélisation académique partielle du contrôle du compensateur d'oscillations (amortisseur hydraulique) et du contrôle du blocage des différenciels sur un tracteur muni d'une remorque (semoir). Notre système simplifié comporte trois processeurs munis d'un exécutif temps réel et reliés par un bus CAN.

Précisons que le modèle que nous présentons est basé sur une architecture matérielle exacte mais sur une spécification logicielle très partielle. Nous nous sommes donc permis d'imaginer, à partir de cette spécification, une architecture logicielle basée sur la présence d'un système d'exploitation temps réel et répondant au cahier des charges.

10.1 Cahier des charges

- Le **processeur 1** est situé à l'avant du tracteur et fournit la commande de blocage des différentiels et la commande de compensation des oscillations du pont avant, en fonction de l'angle de braquage ou de la vitesse d'avancement ;
- Le **processeur 2** est au niveau de la cabine et reçoit les données GPS et les commandes de l'opérateur.
- Le **processeur 3** est sur le semoir et envoie les commandes du vérin hydraulique de l'arbre d'accroche de la remorque ainsi que les commandes d'ouverture de la trappe d'épandage de la semence.

Nous notons $Taski_j$ la i^{eme} tâche du processeur j. Toutes les tâches sont périodiques (de période 40 unités de temps sauf $Task3_1$ et $Task2_2$ dont la période est 20 unités de temps). La modélisation de l'accès au bus CAN est faite par une relation de priorité. Les identifiants des messages sont tels que les messages de $Task1_1$ ont toujours une plus grande priorité que les messages de $Task1_2$.

- Processeur 1
  - $Task1_1$: contrôle le compensateur d'oscillations du tracteur ; elle envoie des messages CAN à $Task1_3$ (tâche 1 sur le Processeur 3) et se synchronise avec $Task2_1$ par un sémaphore ;

  - $Task2_1$: contrôle le blocage des différenciels à partir des dernières mises-à-jour locales de la vitesse et de la commande de direction (le rafraîchissement est effectué par $Task3_1$) et à partir de la commande de compensation d'oscillations (directement fournie par la tâche $Task1_1$, la synchronisation se faisant par l'intermédiaire d'un sémaphore) ;
  - $Task3_1$: rafraîchit les données locales : commande de direction et vitesse. L'obtention de ces données n'est pas représentée ici.

  La relation de priorité sur le Processeur 1 est : $Task2_1 > Task1_1 > Task3_1$
- Processeur 2
  - $Task1_2$: acquiert et transmet les données GPS (position du véhicule) : rafraîchit les données localement et transmet l'information à la remorque (*via* des messages CAN vers $Task2_4$) ;
  - $Task2_2$: acquiert et met à jour les commandes de l'opérateur et le mode de fonctionnement.

  La relation de priorité sur le Processeur 2 est : $Task1_2 > Task2_1$
- Processeur 3
  - $Task1_3$: contrôle le vérin hydraulique de l'arbre d'accroche (attend les messages CAN en provenance de $Task1_1$) ;
  - $Task2_3$: contrôle le vérin hydraulique d'ouverture de la trappe d'épandage de la remorque semoir (attend les messages CAN en provenance de $Task1_2$).

  La relation de priorité sur le Processeur 3 est : $Task1_3 > Task2_3$

10.2 Modélisation

Toutes les tâches sont périodiques. Leurs modèles seront donc tous basés sur la figure 6.3a du chapitre 6. La modélisation du bus CAN est faite selon le motif de la figure 6.8 et celle du sémaphore selon la figure 6.4. La communication *via* un tableau noir (ou *blackboard*) utilisée par exemple entre $Task3_1$ et $Task2_1$ n'est pas représentée explicitement car elle ne contraint pas le système.

La modélisation du système est donnée sur la figure 10.1. Les temps donnés ici sont indicatifs mais pourraient être obtenus par des calculs de temps d'exécutions classiques par analyse statique et dynamique de code (voir par exemple [CPRS03] et - ou - par simulation (voir par exemple [Bri04]).

10.3 Analyse

Nous appliquons la méthode du chapitre 7 à ce modèle et nous obtenons (en moins d'une seconde sur PowerPC G3 à 800Mhz avec 640Mo de RAM) un automate possédant 318 localités, 596 transitions et 7 chronomètres.

Nous nous intéressons maintenant au calcul des temps de réponse de toutes les tâches du système. Pour cela nous utilisons un automate observateur des tirs de transition similaire à ceux décrits dans le chapitre 5.

Il possède une localité et autant de transitions que de transitions dans le réseau de Petri. Nous ajoutons des horloges remises à zéro pour les transitions qui nous intéressent. La description HyTech de cet automate est la suivante :

```
automaton verifier
```

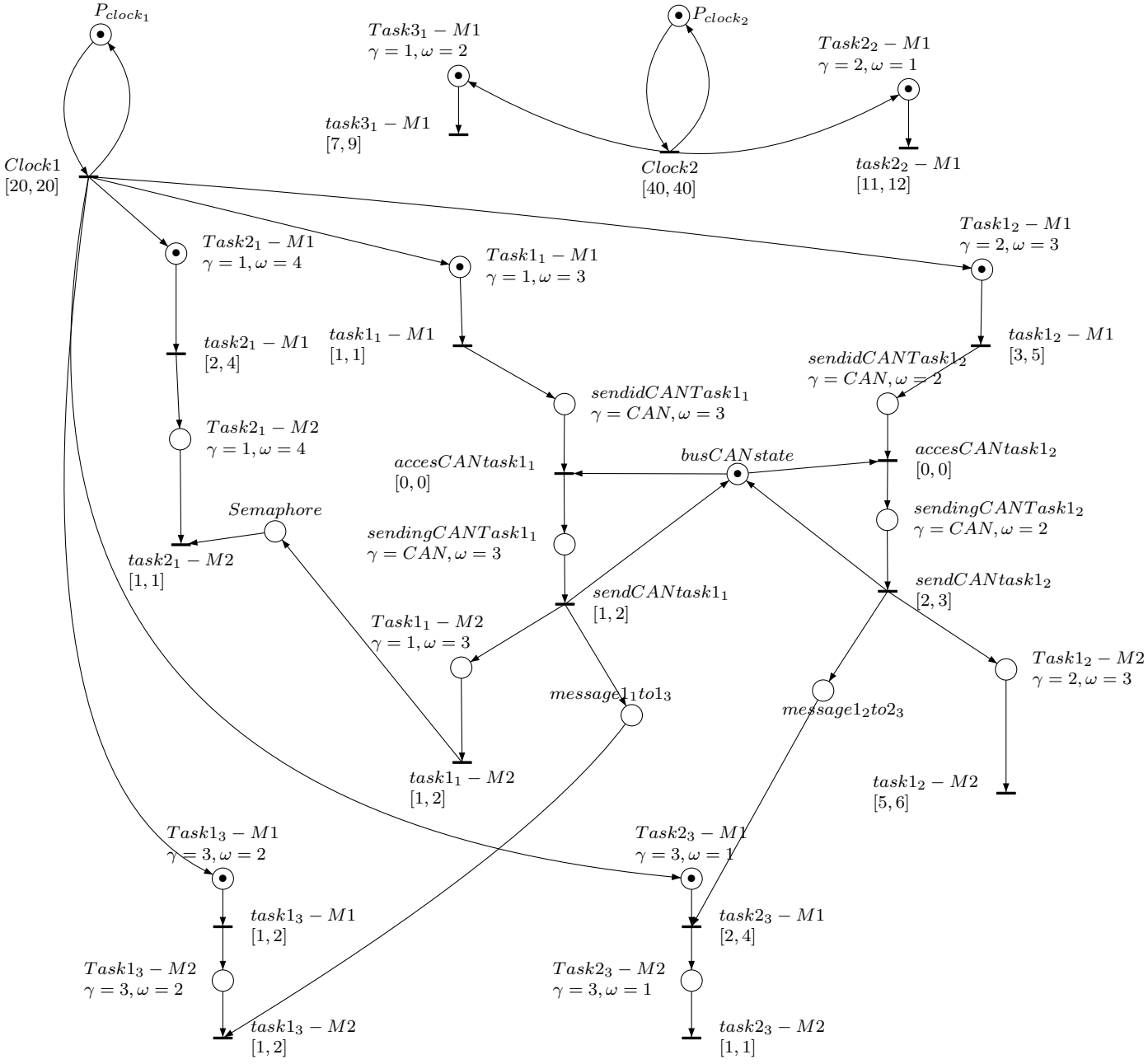

FIG. 10.1 – Modèle *Scheduling*-TPN partiel du système de contrôle du tracteur

```
synclabs:T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11,
T12, T13, T14, T15, T16, T17;
initially C0 & y1 = 0 & y2 = 0;
loc C0: while True wait{}
when True sync T0 do {y1'=0} goto C0; -- Clock1
when True sync T1 goto C0; -- Clock2
when True sync T2 goto C0; -- task3_1-M1
when True sync T3 goto C0; -- task2_2-M1
when True sync T4 goto C0; -- task1_1-M1
when True sync T5 goto C0; -- task1_2-M1
when True sync T6 goto C0; -- accessCANtask1_1
when True sync T7 goto C0; -- sendCANtask1_1
when True sync T8 goto C0; -- sendCANtask1_2
when True sync T9 goto C0; -- accessCANtask1_2
when True sync T10 goto C0; -- task1_1-M2
when True sync T11 goto C0; -- task1_2-M2
when True sync T12 goto C0; -- task2_1-M1
when True sync T13 do {y2'=0} goto C0; -- task2_1-M2
when True sync T14 goto C0; -- task1_3-M1
when True sync T15 goto C0; -- task1_3-M2
when True sync T16 goto C0; -- task2_3-M1
when True sync T17 goto C0; -- task2_3-M2
end -- verifier
```

Tel quel, cet automate observe les tirs de $Clock1$, fournissant l'horloge y_1, et $task2_1 - M2$, fournissant l'horloge y_2. Nous pouvons vérifier que la tâche $Task2_1$ ne dépasse jamais son échéance en vérifiant qu'aucun état n'est tel que $y_1 - y_2 > 20$, ce qui est réalisé par les instructions HyTech suivantes :

```
var
    final_reg, init_reg, reached : region;

init_reg := loc[scswa] = C0
            & loc[verifier] = C0
            & x0 = 0
            & x1 = 0
            & x2 = 0
            & x3 = 0
            & x4 = 0
            & x5 = 0
            & x6 = 0
            & y1 = 0
            & y2 = 0;

final_reg :=  y1 - y2 > 20;
reached := reach forward from init_reg endreach;
print reached & final_reg;
print trace to final_reg using reached;
```

Dans le cas où la tâche dépasserait son échéance, la dernière instruction fournit une trace permettant d'identifier la séquence de tirs responsable du dépassement. Dans la cas présenté ici, nous calculons tout l'espace d'états avant d'en réaliser l'intersection avec la propriété qui nous intéresse. Nous pourrions nous arrêter dès que la propriété est vérifiée. Le calcul de l'espace d'états prend ici environ 20 secondes.

En procédant par dichotomie, nous pouvons calculer, par cette méthode, le pire temps de réponse la tâche (*Worst Case Response Time* ou WCRT). Il s'agit du plus grand entier E tel que l'ensemble des états vérifiant $y_1 - y_2 > E + 1$ soit vide. De la même façon, nous pouvons trouver le meilleur temps de réponse (*Best Case Response Time* ou BCRT) : il est donné par $T - e$ où e est le plus grand entier tel que l'ensemble des états vérifiant $y_2 - y_1 > e + 1$ soit vide et T est la période de la tâche..

Nous obtenons les temps de réponse donnés dans le tableau 10.1.

Processeur	Tâche	BCRT	WCRT
1	1	5	12
1	2	6	13
1	3	13	20
2	1	10	16
2	2	19	20
3	1	5	12
3	2	7	17

TAB. 10.1 – Meilleurs et pires temps de réponse des tâches du système de la figure 10.1

Nous pouvons déduire de ces chiffres que le système est ordonnançable. Par ailleurs, nous pouvons vérifier d'autres propriétés en utilisant la même méthode : par exemples, en observant les transitions $task3_1 - M1$ et $task2_1 - M2$, nous pouvons vérifier que toute écriture du tableau noir sur le processeur 1 est suivie d'une lecture moins de 13 unités de temps plus tard. En particulier, cela signifie qu'aucune donnée n'est jamais périmée car la production (tir de $task3_1 - M1$) s'effectue au mieux toutes les 53 unités de temps.

10.4 Conclusion

Dans ce chapitre, nous avons présenté une courte étude de cas afin d'illustrer l'utilisation pratique du résultat principal de cette thèse : la traduction d'un réseau de Petri T-temporel étendu à l'ordonnancement en un automate à chronomètres. En particulier, nous avons montré comment calculer des temps de réponse exacts pour des systèmes complexes incluant une architecture distribuée et des durées d'exécution variables.

Il serait bien sûr intéressant d'appliquer les techniques développées dans cette thèse à une étude de cas plus complète issue de l'industrie.

Cinquième partie

Conclusion

Chapitre 11

Conclusions et perspectives

Bilan

Avec l'essor des systèmes temps réel, en nombre, en importance mais aussi en taille et en complexité, leur vérification devient un enjeu de premier ordre. En particulier, des modèles plus proches du comportement réel des applications sont nécessaires ainsi que des techniques permettant de vérifier des propriétés temporelles quantitatives.

Cette thèse apporte des nouvelles contributions dans ces deux directions. En effet nous montrons comment modéliser et vérifier les applications temps réel avec une extension des réseaux de Petri T-temporels prenant en compte la politique d'ordonnancement de l'exécutif (*Scheduling*-TPN). Ce modèle met en œuvre le concept de chronomètre (*stopwatch*), c'est-à-dire d'horloge pouvant être arrêtée et redémarrée avec la mémoire du temps écoulé avant son arrêt.

Nous avons prouvé que les problèmes « intéressants », tels que l'accessibilité d'états, sont indécidables pour ce modèle, même pour des réseaux bornés.

Nous avons proposé des semi-algorithmes de calcul de l'espace d'états. Pour l'élaboration de ces semi-algorithmes, nous avons d'abord étudié le cas, plus simple, des modèles temporisés et plus particulièrement des réseaux de Petri T-temporels de Merlin. Pour ces derniers, nous savons que les problèmes intéressants liés à la vérification sont décidables pour des réseaux bornés. Par ailleurs, la méthode du graphe des classes atomiques permet de vérifier des propriétés de type CTL^*.

Afin de vérifier des propriétés temporelles *quantitatives*, nous avons proposé une traduction, basée sur le calcul du graphe des classes d'états, d'un réseau de Petri T-temporel borné en un automate temporisé qui lui est temporellement bisimilaire. À partir de cette traduction, nous avons montré comment exprimer des propriétés $TCTL$ sur le réseau de Petri T-temporel et les vérifier efficacement, grâce à un algorithme de réduction du nombre d'horloges à la volée lors de la traduction, sur l'automate temporellement bisimilaire avec des outils comme Uppaal ou Kronos. Cela montre en outre que $TCTL$ est décidable pour les réseaux de Petri T-temporels bornés.

Dans le cadre des modèles à chronomètres, nous avons proposé un calcul exact de l'espace d'états d'un *Scheduling*-TPN étendant naturellement le graphe des classes d'états des réseaux de Petri T-temporels et mettant en œuvre des polyèdres généraux. Nous en avons également proposé une surapproximation à base de DBM, plus rapide à calculer.

En utilisant ce calcul, nous avons adapté notre méthode de traduction afin de pouvoir

traduire les *Scheduling*-TPN bornés en automates à chronomètres. Cette traduction bénéficie d'un mécanisme de réduction du nombre de chronomètres à la volée similaire à celui du cas temporisé et surtout, nous montrons qu'elle peut se faire en utilisant un algorithme efficace à base de DBM, l'automate résultant étant temporellement bisimilaire au *Scheduling*-TPN initial. Cette méthode nous permet de vérifier des systèmes temps réel bien plus complexes qu'une modélisation directe sous la forme d'un produit d'automates à chronomètres.

Enfin, nous avons montré comment étendre la classe des politiques d'ordonnancement modélisables par des *Scheduling*-TPN afin de pouvoir prendre en compte les politiques *Round-Robin* et *Earliest Deadline First*. Nous avons également montré comment adapter le calcul du graphe des classes d'états pour ces deux politiques.

Perspectives

Dans la continuité de ce travail un certain nombre de problèmes restent à étudier. À court terme, nous pensons étendre le calcul de l'automate à chronomètres aux politiques d'ordonnancement *Round-Robin*[1] et *Earliest Deadline First*. Par ailleurs, le calcul du graphe des classes d'états pour ces deux politiques d'ordonnancement reste à implémenter.

De façon moins immédiate, il pourrait être intéressant d'étudier l'approximation des polyèdres par des techniques moins grossières que les DBM. De nombreux travaux existent dans le cas des automates hybrides, à partir desquels nous pourrions aborder le problème, par exemple [CK99, GM99, SK00, Bot00, KV01, ADM02]. L'adaptation au calcul du graphe des classes d'états ne semble cependant pas immédiate.

À plus long terme, les approches impliquant une sémantique à temps discret paraissent prometteuses grâce aux progrès réalisés dans le calcul et la réprésentation symbolique d'espaces d'états de très grande taille pour les réseaux de Petri non temporisés [MDR02, Cia04]. Cependant, de telles sémantiques fournissent des espaces d'états qui sont des sous-approximations (donc non conservatives) de l'espace d'états obtenu avec un temps dense, ce qui engendre de nouveaux problèmes.

[1] Dans le cas de *Round-Robin*, la traduction se fera sans doute plus naturellement vers un automate hybride linéaire.

Bibliographie

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1) :2–34, may 1993.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138 :3–34, january 1995.

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

[ADM02] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *Proceedings of the 14th International Conference on Computer Aided Verification*, pages 365–370. Springer-Verlag, 2002.

[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.

[AGS02] K. Altisen, G. Gössler, and J. Sifakis. Scheduler modelling based on the controller synthesis paradigm. *Journal of Real-Time Systems*, 23 :55–84, 2002. Special issue on control-theoritical approaches to real-time computing.

[AH94] R. Alur and T.A. Henzinger. A really temporal logic. *Journal of the Association for Computing Machinery*, 41(1) :181–204, 1994.

[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.

[AlP+99] K. Altisen, G. Gößler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *20th IEEE Real-Time Systems Symposium (RTSS'99)*, pages 154–163, Phoenix, Arizona, USA, december 1999. IEEE Computer Society Press.

[AlS00] K. Altisen, G. Gößler, and J. Sifakis. A methodology for the construction of scheduled systems. In *6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00)*, volume 1926 of *Lecture Notes in Computer Science*, pages 106–120, Pune, India, september 2000. Springer-Verlag.

[AN01] P.A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–72, Newcastle upon Tyne, United Kingdom, june 2001. Springer-Verlag.

[Arn94] A. Arnold. *Finite Transition System.* Prentice Hall, 1994.

[BCvH+03] J. Billington, S. Christensen, K.M. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. The petri net markup language : Concepts, technology, and tools. In Wil M. P. van der Aalst and Eike Best, editors, *24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 483–505, Eindhoven, The Netherlands, June 2003. Springer-Verlag.

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE transactions on software engineering*, 17(3) :259–273, 1991.

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

[BFSV03] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Modeling flexible real time systems with preemptive time Petri nets. In *15th Euromicro Conference on Real-Time Systems (ECRTS'2003)*, pages 279–286, 2003.

[BFSV04] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Time state space analysis of real-time preemptive systems. *IEEE transactions on software engineering*, 30(2) :97–111, February 2004.

[BLRV04] B. Berthomieu, D. Lime, O.H. Roux, and F. Vernadat. Reachability problems and abstract state spaces for time Petri nets with stopwatches. Technical Report 04483, Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS), Toulouse, France, October 2004.

[BM83] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. *IFIP Congress Series*, 9 :41–46, 1983.

[BM03] H. Boucheneb and J. Mullins. Analyse des réseaux temporels. calcul des classes en o(n2) et des temps de chemin en o(m*n). *Techniques et Sciences Informatiques*, 22(4) :277–299, 2003.

[BMR90] S. Baruah, A. Mok, and L. Rosier. Preemtively scheduling hard-real-time sporadic tasks on one processor. In *11th Real-Time Systems Symposium*, pages 182–1990, 1990.

[Bot00] Oleg Botchkarev. Ellipsoidal techniques for verification of hybrid systems. 2000.

[Bri04] Mikaël Briday. *Validation par simulation fine d'une architecture opérationnelle.* PhD thesis, Université de Nantes, december 2004.

[BST98] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. *Lecture Notes in Computer Science*, 1536 :103–129, 1998.

[BT01] E. Brinksma and J. Tretmans. Testing transition systems. In *Modelling and Verification of Parallel Processes (MOVEP2k)*, volume 2067 of *Lecture Notes in Computer Science*, pages 187–195, Nantes, France, 2001. Springer-Verlag.

[BV03] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003)*, pages 442–457. Springer–Verlag, Apr 2003.

[Čer92] K. Čerāns. *Algorithmic problems in analysis of real time system specifications.* University of Latvia, Dr.sc.comp. Thesis, 1992.

[Cia04] G. Ciardo. Reachability set generation for petri nets : Can brute force be smart ? In *The 25th International Conference on Application and Theory of Petri nets, (ICATPN 2004)*, volume 3099 of *Lecture Notes in Computer Science*, pages 17–34, Bologna, Italy, june 2004. Springer-Verlag.

[CK99] Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. *Lecture Notes in Computer Science*, 1569 :76– ? ?, 1999.

[CL00] Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In Catuscia Palamidesi, editor, *11th International Conference on Concurrency Theory, (CONCUR'2000)*, number 1877 in Lecture Notes in Computer Science, pages 138–152, University Park, P.A., USA, July 2000. Springer-Verlag.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. Cambridge : MIT Press, 1990.

[Cof76] E.G. Coffman. Introduction to deterministic scheduling theory. *Computer and Job-Shop Scheduling Theory*, pages 1–50, 1976.

[CPRS03] A. Colin, I. Puaut, C. Rochange, and P. Sainra. Calcul de majorants de pire temps d'exécution : état de l'art. *Techniques et Sciences Informatiques (TSI)*, 22(5) :651–677, 2003.

[CR03] F. Cassez and O.H. Roux. Traduction structurelle des réseaux de Petri temporels vers les automates temporisés. In *4ième Colloque Francophone sur la Modélisation des Systèmes Réactifs, (MSR'03)*, Metz, France, october 2003.

[CR04] F. Cassez and O.H. Roux. Structural translation from time petri nets to timed automata. In *The 4th International Workshop on Automated Verification of Critical Systems (AVoCS 2004)*, London, United Kingdom, september 2004. to appear.

[Dan63] G.B. Dantzig. Linear programming and extensions. *IEICE Transactions on Information and Systems*, 1963.

[Der74] M.L. Dertouzos. Control robotics : the procedural control of physical processes. *Information Processing*, 1974.

[dFRA00] D. de Frutos Escrig, V. Valero Ruiz, and O. Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*, volume 1825 of *Lecture Notes in Computer Science*, pages 187–206, Aarhus, Denmark, june 2000. Springer-Verlag.

[Dij72] E. W. Dijkstra. Notes on structured programming. In O. J. Dahl et al., editor, *Structured Programming*, pages 1–82. Academic Press, London, 1972.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Workshop Automatic Verification Methods for Finite-State Systems*, volume 407, pages 197–212, 1989.

[DS94] M. Diaz and P. Senac. Time stream Petri nets : a model for timed multimedia information. *Lecture Notes in Computer Science*, 815 :219–238, 1994.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *1996 IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81, Washington, DC, USA, december 1996. IEEE Computer Society Press.

[Fer89] J.-C. Fernandez. Aldebaran : A tool for verification of communicating processes. Technical report, Institut d'Informatique et Mathématiques Appliquées de Grenoble, 1989.

[FMPY03] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis using two clocks. In Hubert Garavel and John Hatcliff, editors, *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*, volume 2619 of *Lecture Notes in Computer Science*, pages 224–239. Springer-Verlag, April 2003.

[FPY02] E. Fersman, P. Petterson, and W. Yi. Timed automata with asynchronous processes : Schedulability and decidability. In *8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82, Grenoble, France, 2002. Springer-Verlag.

[Gal97] L. Gallon. *Le modèle réseaux de Petri temporisés stochastiques : extensions et applications.* PhD thesis, Université Paul Sabatier, Toulouse, France, 1997.

[GM99] M. R. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. *Lecture Notes in Computer Science*, 1569 :103–116, 1999.

[gro01] OSEK group. *OSEK/VDX specification.* http ://www.osek-vdx.org, september 2001.

[GRR03] G. Gardey, O.H. Roux, and O.F. Roux. A zone-based method for computing the state space of a time Petri net. In *In Formal Modeling and Analysis of Timed Systems, (FORMATS'03)*, volume LNCS 2791. Springer-Verlag, September 2003.

[GRR05] G. Gardey, O.H. Roux, and O.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.

[HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech : A model checker for hybrid systems. *Journal of Software Tools for Technology Transfer*, 1(1-2) :110–122, 1997.

[HKL91] M.G. Harbour, M.H. Klein, and J.P. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *12th IEEE Real-Time Systems Symposium (RTSS'91)*, pages 116–128, San Antonio, USA, december 1991. IEEE Computer Society Press.

[HKPV98] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata ? *Journal of Computer and System Sciences*, 57 :94–124, 1998.

[HKR01] W. Henderson, D. Kendall, and A. Robson. Improving the accuracy of scheduling analysis applied to distributed systems. *Real-Time Systems*, 20(1) :5–25, 2001.

[Hla04] P.-E. Hladik. *Analyse d'ordonnanςbilité et méthode de placement pour la conception d'architectures opérationnelles de systèmes temps réel préemptifs à priortés fixes en environnement distribué.* PhD thesis, Université de Nantes, 2004.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2) :193–244, 1994.

[JLL77] N.D. Jones, L.H. Landweber, and Y.E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science 4*, pages 277–299, 1977.

[Jua99] G. Juanole. Modélisation et évaluation du protocole mac du réseau can. In *Rapport LAAS No99303. Ecole d'Été Applications, Réseaux et Systémes (ETR'99)*, pages 187–200, Poitiers, France, september 1999.

[KDC96] W. Khansa, J.-P. Denat, and S. Collart-Dutilleul. P-Time Petri Nets for manufacturing systems. In *International Workshop on Discrete Event Systems, WODES'96*, pages 94–102, Edinburgh (U.K.), august 1996.

[KV01] Alexander B. Kurzhanski and Pravin Varaiya. On ellipsoidal techniques for reachability analysis. part i. external approximations. In *Optimization Methods and Software*, volume 17, pages 177–206, 2001.

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE transactions on software engineering*, 3(2) :125–143, 1977.

[Lil99] J. Lilius. Efficient state space search for time Petri nets. In *MFCS Workshop on Concurrency '98*, volume 18 of *ENTCS*. Elsevier, 1999.

[LL73] C. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, 20(1) :44–61, january 1973.

[LPY95] K.G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Fundamentals of Computation Theory*, pages 62–88, 1995.

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1–2) :134–152, Oct 1997.

[LR] D. Lime and O.H. Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Events Dynamic Systems (DEDS)*. Accepté avec modifications mineures.

[LR03a] D. Lime and O.H. Roux. Expressiveness and analysis of scheduling extended time Petri nets. In *5th IFAC International Conference on Fieldbus Systems and their Applications, (FET 2003)*, Aveiro, Portugal, July 2003. Elsevier Science.

[LR03b] D. Lime and O.H. Roux. State class timed automaton of a time Petri net. In *10th International Workshop on Petri Nets and Performance Models, (PNPM 2003)*. IEEE Computer Society, September 2003.

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

[LW82] J.Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 22 :237–250, 1982.

[MD78] A.K. Mok and M. Dertouzos. Multiprocessor scheduling in a hard real-time environment. In *Seventh Texas Conf. on Computing Systems*, 1978.

[MDR02] P. Molinaro, D. Delfieu, and O.H. Roux. Improving the calculus of the marking graph of Petri nets with bdd-like structures. In *IEEE international conference on systems, man and cybernetics (SMC 2002)*, Hammamet, Tunisia, october 2002.

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

[Mer74] P.M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1974.

[Min61] M. Minsky. Recursive unsolvability of Post's problem. *Ann. of Math*, 74 :437–454, 1961.

[MV94] J. McManis and P. Varaiya. Suspension automata : A decidable class of hybrid automata. In David L. Dill, editor, *6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 105–117, Stanford, CA, USA, June 1994. Springer-Verlag.

[OY96] Y. Okawa and T. Yoneda. Schedulability verification of real-time systems with extended time Petri nets. *International Journal of Mini and Microcomputers*, 18(3) :148–156, 1996.

[PH98] J.C. Palencia and M.G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 26–37, Madrid, Spain, december 1998. IEEE Computer Society Press.

[PH99] J.C. Palencia and M.G. Harbour. Exploiting precedence relations in the scheduling analysis of distributed real-time systems. In *20th IEEE Real-Time Systems Symposium (RTSS'99)*, pages 328–339, Phoenix, Arizona, USA, december 1999. IEEE Computer Society Press.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th IEEE Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[PT99] M. Pezze and M. Toung. Time Petri nets : A primer introduction. Tutorial presented at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications, Zaragoza, Spain, september 1999.

[Ram74] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974. Project MAC Report MAC-TR-120.

[RD02] O. H. Roux and A.-M. Déplanche. A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation (JESA)*, 36(7), 2002.

[RL04] O.H. Roux and D. Lime. Time Petri nets with inhibitor hyperarcs. Formal semantics and state space computation. In *The 25th International Conference on Application and Theory of Petri Nets, (ICATPN 2004)*, volume 3099 of *Lecture Notes in Computer Science*, pages 371–390, Bologna, Italy, June 2004. Springer-Verlag.

[Rus01] J. M. Rushby. Theorem proving for verification. In *Modelling and Verification of Parallel Processes (MOVEP2k)*, volume 2067 of *Lecture Notes in Computer Science*, pages 39–57, Nantes, France, 2001. Springer-Verlag.

[Sav01] A.T. Sava. *Sur la synthèse de la commande des systèmes à évènements discrets temporisés*. PhD thesis, Institut National polytechnique de Grenoble, Grenoble, France, november 2001.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, NY,, 1986.

[SK00] B. Silva and B. H. Krogh. Formal verification of hybrid systems using checkmate : A case study. In *Proceedings of the American Control Conference*, pages 1679–1683, Chicago, IL, 2000.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols : An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9) :1175–1185, 1990.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems (extended abstract). In *13th Syposium on Theoretical Aspects of Computer Science*, pages 347–359, Grenoble, France, february 1996. Springer-Verlag.

[TC94] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(1-2) :117–134, 1994.

[Tin94] K. Tindell. *Fixed priority scheduling of hard real-time systems*. PhD thesis, Department of Computer Science, University of New York, 1994.

[TSLT97] J. Toussaint, F. Simonot-Lion, and Jean-Pierre Thomesse. Time constraint verification methods based on time Petri nets. In *6th Workshop on Future Trends in Distributed Computing Systems (FTDCS'97)*, pages 262–267, Tunis, Tunisia, 1997.

[Vin03] Aymeric Vincent. *Conception et réalisation d'un vérificateur de modèles Alta-Rica*. PhD thesis, Université of Bordeaux 1, December 2003.

[Yov97] S. Yovine. Kronos : A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1–2) :123–133, Oct 1997.

[YR98] T. Yoneda and H. Ryuba. CTL model checking of time Petri nets using geometric regions. *IEICE Transactions on Information and Systems*, E99-D(3) :297–396, march 1998.

Annexe A

Preuve du théorème 22

Théorème 22 (Valeur des observateurs) *Soit* $\rho = s_1 = (M_1, \nu_1) \overset{t_1,d_1}{\rightarrow} s_2 \overset{t_2,d_2}{\rightarrow} \cdots \overset{t_{k-1},d_{k-1}}{\rightarrow} s_k$ *une séquence de tir d'un Scheduling-TPN* $\mathcal{T} = (P, T, {}^\bullet(), ()^\bullet, \alpha, \beta, M_0, Flow)$. *Soit* $t_i \in T - \{t_1, \ldots, t_k\}$ *une transition observatrice de* ρ. *Alors* $\Omega - \beta_i^{(k)} \leq d_1 + d_2 + \cdots + d_{k-1} \leq \Omega - \alpha_i^{(k)}$.

Preuve. La preuve repose sur l'algorithme de calcul des domaines des classes. Considérons une classe $C = (M, D)$ et une transition tirable t_f.

$$D = P(\theta_0, \ldots, \theta_{f-1}, \theta_f, \theta_{f+1}, \ldots, \theta_n) \tag{A.1}$$

P étant un polyèdre.

Pour calculer le domaine de la classe $C' = (M', D')$ obtenue par le tir de t_f depuis C, nous effectuons les étapes suivantes

1. effectuons les changements de variables pour les transitions actives : soient $t_0, \ldots, t_n$ les transitions sensibilisées et $\theta_i, i \in [0, n]$ les variables de D les représentant. Supposons que $t_0, \ldots, t_m$ sont actives. Alors :

$$\forall i \in [\![0, m]\!], \theta_i = \theta'_i + \theta_f \tag{A.2}$$

Nous avons donc le nouveau domaine $D^{(1)}$:

$$D^{(1)} = \begin{cases} P(\theta'_0 + \theta_f, \ldots, \theta'_{f-1} + \theta_f, \theta_f, \theta'_{f+1} + \theta_f, \ldots, \theta'_n + \theta_f) \\ \theta_0 = \theta'_0 + \theta_f \\ \vdots \\ \theta_m = \theta'_m + \theta_f \\ \theta_{m+1} = \theta'_{m+1} \\ \vdots \\ \theta_n = \theta'_n \end{cases} \tag{A.3}$$

2. éliminons θ_f de $D^{(1)}$ par la méthode de Fourier-Motzkin [Dan63]. Soit $D^{(2)}$ le système

résultant :

$$D^{(2)} = \begin{cases} P(\theta'_0, \ldots, \theta'_{f-1}, \theta'_{f+1}, \ldots, \theta'_n) \\ \theta_0 = \theta'_0 + \theta_f \\ \vdots \\ \theta_m = \theta'_m + \theta_f \\ \theta_{m+1} = \theta'_{m+1} \\ \vdots \\ \theta_n = \theta'_n \end{cases} \tag{A.4}$$

3. ajoutons les contraintes $\forall i \in [\![0, n]\!], \theta'_i \geq 0$ ce qui nous donne le système $D^{(3)}$.

$$D^{(3)} = \begin{cases} P(\theta'_0, \ldots, \theta'_{f-1}, \theta'_{f+1}, \ldots, \theta'_n) \\ \theta'_0 = \max\{0, \theta_0 - \theta_f\} \\ \vdots \\ \theta'_m = \max\{0, \theta_m - \theta_f\} \\ \theta'_{m+1} = \theta_{m+1} \\ \vdots \\ \theta'_n = \theta_n \end{cases} \tag{A.5}$$

4. finalement, nous éliminons les variables relatives aux transitions désensibilisées par le nouveau marquage M' et nous ajoutons les inéquations relatives aux transitions nouvellement sensibilisées par M'.

Considérons maintenant la séquence de tir $s_0 \overset{t_0}{\rightarrow} \ldots \overset{t_{k-1}}{\rightarrow} s_k$ menant à la classe $C = (M, D)$. Supposons qu'une transition t_i était nouvellement sensibilisée dans s_0 et est restée active durant l'intégralité de la séquence de tir. Puisque t_i était active dans la classe précédente, nous avons

$$\theta_i^{(k)} = \max\{0, \theta_i^{(k-1)} - \theta_{k-1}^{(k-1)}\} \tag{A.6}$$

Étant donné que t_i est restée active durant l'intégralité de la séquence de tir, nous avons, par une simple récurrence :

$$\theta_i^{(k)} = \max\{0, \max\{0, \max\{\cdots\} - \theta_{k-2}^{(k-2)}\} - \theta_{k-1}^{(k-1)}\} \tag{A.7}$$

De plus, puisque $\alpha(t_i) \geq \sum_{j \in [\![0,k-1]\!]} \beta(t_j)$, nous savons que $\forall j \in [\![0, k-1]\!], \theta_i^{(j)} - \theta_j^{(j)} \geq 0$. Donc, nous avons :

$$\theta_i^{(k)} = \theta_i^{(0)} - \theta_0^{(0)} - \cdots - \theta_{k-2}^{(k-2)} - \theta_{k-1}^{(k-1)} \tag{A.8}$$

Ce que nous pouvons réécrire en :

$$\theta_i^{(0)} - \theta_i^{(k)} = \theta_0^{(0)} + \cdots + \theta_{k-2}^{(k-2)} + \theta_{k-1}^{(k-1)} \tag{A.9}$$

Nous connaissons l'intervalle de valeurs $[\alpha_i^{(k)}, \beta_i^{(k)}]$ de $\theta_i^{(k)}$ en résolvant partiellement $D^{(k)}$. Nous savons aussi que $\theta_i^{(0)} = \Omega$. Et donc finalement :

$$\Omega - \beta_i^{(k)} \leq \theta_0^{(0)} + \cdots + \theta_{k-2}^{(k-2)} + \theta_{k-1}^{(k-1)} \leq \Omega - \alpha_i^{(k)} \tag{A.10}$$

□