Vérification et dépliages de réseaux de Petri temporels paramétrés

Louis-Marie Traonouez

UNIVERSITÉ DE NANTES

ÉCOLE DOCTORALE

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATHEMATIQUES

Année : 2009

Thèse de Doctorat de l'Université de Nantes

Spécialité : AUTOMATIQUE ET INFORMATIQUE APPLIQUÉE

Présentée et soutenue publiquement par :

Louis-Marie Traonouez

le 27 novembre 2009

à l'École Centrale de Nantes

TITRE
Vérification et dépliages de réseaux de Petri temporels paramétrés

Jury

Président :	Jean-Louis BOIMOND	Professeur à l'ISTIA, LISA, Angers
Rapporteurs :	Claude JARD	Professeur à l'ENS Cachan, IRISA, Rennes
	François LAROUSSINIE	Professeur à l'Université Paris Diderot, LIAFA, Paris
Examinateurs :	Bernard BERTHOMIEU	Chargé de recherches CNRS, LAAS, Toulouse
	David DELFIEU	Maître de conférence à l'École Polytechnique de Nantes, IRCCyN
	Olivier (H.) ROUX	Maître de conférence HDR à l'IUT de Nantes, IRCCyN

Directeur de thèse : **Olivier (H.) ROUX**
Laboratoire / composante : IRCCyN / IUT de Nantes
Co-encadrant : **David DELFIEU**
Laboratoire / composante : IRCCyN / École Polytechnique de Nantes

ED 503-075

Remerciements

Je vais essayer de remercier l'ensemble des personnes qui ont permis l'aboutissement de ces recherches.

En premier lieu, je souhaite remercier Claude Jard et François Laroussinie qui me font l'honneur d'être les rapporteurs de cette thèse, ainsi que Bernard Berthomieu et Jean-Louis Boimond pour leur participation au jury de thèse. Je tiens à remercier tout particulièrement Claude Jard pour sa collaboration sur le thème des dépliages : son expertise du domaine fut une aide précieuse.

Je remercie ensuite chaleureusement mon directeur de thèse Olivier Roux. Pendant ces trois années il a su me guider sur le chemin adéquat. Je lui suis gré de l'enthousiasme qu'il a manifesté pour mes recherches (même lorsque le mien faillissait), et des réponses qu'il a apporté à mes nombreuses interrogations.

Après quatre années de travail ensemble, je tiens à remercier pleinement David Delfieu. C'est avec lui que j'ai découvert la recherche lors de mon master. Je peux maintenant apprécier le chemin que nous avons parcouru.

Je lui dois également une grande partie de ma réussite tant son aide scientifique a comptée pour moi. Pour cela, pour la collaboration enrichissante que nous avons eue et pour son amitié, je remercie du fond du coeur Didier Lime.

Merci à Yvon Trinquet qui m'a accueilli dans son équipe, aux membres de l'équipe temps réel que j'ai eu le plaisir de côtoyer depuis mon master, et à tout le personnel du laboratoire grâce auquel j'ai bénéficié d'un environnement de travail des plus enviables.

Une thèse c'est une aventure très personnelle, parfois solitaire même. Heureusement nous sommes encore nombreux à l'entreprendre, et ceci m'a permis de rencontrer des personnes formidables. Merci donc à tous ces thésards (ou ancien thésards) avec qui j'ai partagé cette aventure. Merci à Charlotte, Rola, Di, Jonathan et Médésu : nous avons commencé notre thèse ensemble et nous avons la chance de la terminer ensemble. Merci à Pedro et Loïc pour ces soirées barbecue qui sont venues égayer une dernière année chargée. Merci enfin à Emilie, Clément, Jamil, Jordan, Paul-André, Morgan, Isabelle, et tous ceux que hélas j'oublie. Merci également aux associations de doctorants qui animent et font évoluer le milieu de la recherche, et permettent à des doctorants de différents univers de se rencontrer : l'AED, LOGIN, la CJCN.

Pour terminer je remercie ma famille et mes amis. À chaque fois que j'ai pu les retrouver cela m'a procuré des instants de repos salutaires. Merci en particulier à mes parents qui m'ont continuellement encouragé. Enfin, je ne remercierai jamais assez mes grands-parents qui m'ont soutenu pendant toutes mes années d'études. Merci à ma grand-mère pour toutes ses attentions, et à mon grand-père pour m'avoir montré la voie à suivre : les moments que j'ai passés petit dans son laboratoire, au milieu de ses expériences, ne sont certainement pas étrangers à ma vocation.

Table des matières

Table des figures

Première partie

Introduction

Chapitre 1

Introduction

Le développement croissant des systèmes informatiques suit plusieurs directions. On assiste d'une part à une diversification de la nature de ces systèmes : ils sont présents dans nos téléphones portables, dans les systèmes embarqués des automobiles, ou bien pour gérer les systèmes d'informations des banques ou encore le réseau global Internet. D'autre part ces systèmes sont de plus en plus interconnectés, que ce soit dans des systèmes distribués ou entre des systèmes hétérogènes : ainsi les différents calculateurs embarqués dans une voiture communiquent entre eux par l'intermédiaire d'un réseau embarqué, mais il y a également une demande pour des applications qui seraient reliées à des réseaux externes tels que l'Internet.

Ces évolutions posent de nombreux défis pour la conception des systèmes informatiques. Il faut notamment être capable de faire interagir des systèmes possédant des exigences très différentes. Par exemple, les systèmes temps réel, que l'on trouve en particulier embarqués dans les systèmes de pilotage des automobiles et des avions, doivent vérifier des contraintes temporelles strictes. Ce n'est par contre pas le cas d'un réseau de communication tel qu'Internet. Lors de la conception du système, ces possibilités de communications externes doivent être prises compte, même si leurs spécifications sont incomplètes ; on doit notamment intégrer le fait qu'elles peuvent constituer des sources d'erreurs. Par ailleurs, la complexité croissante des systèmes amène à les décomposer en plusieurs modules. Chacun des modules est plus simple que le système complet et peut être conçu séparément. Mais en contrepartie, il est parfois difficile de posséder une vue complète du système. La conception est donc réalisée module par module, avant une phase d'intégration qui peut être complexe, notamment lorsque l'on doit s'assurer que les spécifications globales du système sont vérifiées.

Dans le processus de conception d'un système informatique les étapes de vérification sont primordiales. Il s'agit de s'assurer que le système ne connaît pas de défaillances (des comportements non attendus) et qu'il remplisse correctement ses spécifications. Le temps de travail dépensé dans le processus de vérification est en général au moins aussi important que celui passé à vraiment concevoir le système. Les méthodes utilisées pour vérifier un système informatique sont principalement basées sur l'exécution de tests. Des scénarios de tests sont déterminés par les concepteurs, de façon plus ou moins aléatoire, en essayant de couvrir au mieux l'ensemble des sources d'erreurs possibles. On peut également avoir recours à des méthodes de génération automatique de tests. Il sera toutefois impossible de couvrir par des tests l'ensemble des sources d'erreurs. Sur un système complexe, des tests unitaires sont tout d'abord réalisés sur les éléments de bases, puis des tests d'intégrations sont utilisés pour vérifier le bon comportement des modules en interactions. Mais on doit essayer de détecter le

plus tôt possible les erreurs de conceptions afin de diminuer la complexité et le coût de leur correction.

1.1 Systèmes temps réel

Les *systèmes temps réel* sont une classe de systèmes informatiques présentant certaines caractéristiques propres. Ce sont des systèmes informatiques chargés de piloter un procédé physique, en réagissant à l'évolution dynamique de ce procédé : les résultats des programmes sont fonction de données produites par l'environnement (le procédé à contrôler). Pour cette raison on parle également de *systèmes réactifs*. Les temps de réactions du système sont donc contraints par l'évolution dynamique du procédé. Ces systèmes sont présents dans de nombreux domaines : les systèmes de production, les systèmes de pilotage embarqués (automobile, avionique, espace ...), des applications médicales, mais aussi dans les systèmes de gestion des données boursières ou les services multimédias en ligne. Beaucoup de ces applications sont critiques et ne tolèrent pas de défaillance du système informatique de contrôle, ce qui pourrait avoir des conséquences pour la sécurité des usagers.

Pour la conception d'un système temps réel, on commence en général par déterminer l'*architecture logicielle* du système à partir des contraintes fonctionnelles issues du cahier des charges. Elle se compose d'un ensemble de tâches ou de modules pouvant interagir entre eux. On définit ensuite l'*architecture support* du système, qui est constituée des ressources matérielles, telles que les processeurs et les réseaux de communication, ainsi que des supports d'exécution temps réel chargés de gérer l'exécution des tâches et leurs accès aux ressources. La projection de l'architecture logicielle sur l'architecture support fournit l'*architecture opérationnelle* et constitue le système de pilotage temps réel. La vérification des contraintes temporelles dues au temps réel est alors effectuée sur le système global définit par l'architecture opérationnelle. Il s'agit en particulier de s'assurer que les tâches ne dépassent pas leur échéance. Si ce n'est pas le cas la conception du système doit être revue en redéfinissant les architectures logicielle et matérielle.

1.2 Vérification formelle

La conception d'un système informatique commence par la rédaction d'un cahier des charges dans lequel doivent être écrites toutes les spécifications du système. Dès cette étape des problèmes de conception peuvent apparaître : les spécifications sont en effet susceptibles d'être incomplètes, voire contradictoires. La rédaction du cahier des charges en langage naturel peut également être à la source d'erreurs : les spécifications écrites peuvent être ambiguës et mal interprétées par les développeurs chargés de mettre en œuvre des solutions. Enfin, la phase de conception, qui peut être éventuellement découpée en plusieurs étapes mais doit aboutir à la réalisation des programmes informatiques, est évidemment elle-même une source potentielle d'erreurs. Les concepteurs du système sont bien entendu chargés de détecter les erreurs introduites au cours de chacune de ces étapes. Cependant, la complexité croissante des systèmes et le fait qu'il y ait de plus en plus d'interactions avec d'autres systèmes, ou avec des parties du système comme cela est le cas dans les systèmes distribués, rend la logique du système difficile à appréhender pour une personne. La détection des erreurs est donc de plus en plus ardue.

Pour obtenir un système sûr, entre le cahier des charges du système rédigé en langage naturel, et le programme informatique rédigé dans un langage de programmation, il est nécessaire d'utiliser des méthodes claires et précises (c'est-à-dire mathématiques) permettant de faire le lien entre les spécifications informelles, et le progamme final qui est lui-même un objet mathématique. C'est l'objectif des *méthodes formelles* : déterminer des spécifications formelles du système à partir du cahier des charges, et vérifier que les programmes réalisés répondent à ces spécifications. Elles sont classées en trois grandes catégories.

Le **test** est la méthode la plus utilisée pour vérifier un logiciel. Les travaux de recherche dans ce domaine consistent à générer de manière automatique les scénarios de tests à partir d'une spécification formelle du système [Brinksma 01]. Cette méthode est cependant limitée par l'impossibilité de tester de manière exhaustive le système. Elle est cependant la méthode la moins coûteuse et la plus facile à mettre en œuvre, et donc la plus répandue.

La **démonstration automatique** consiste à appliquer des règles de déductions sur une spécification logique du système, afin de prouver une propriété écrite dans cette même logique [Rushby 01]. La décidabilité du problème est cependant limitée par la logique utilisée. En pratique les preuves doivent en général être assistées par un utilisateur compétent.

Enfin, la **vérification de modèles**, ou *model-checking*, nécessite au préalable de construire un modèle formel $\mathcal{S}$ du système, et de déterminer une propriété φ à vérifier, exprimée dans une logique adaptée. Pour vérifier que $\mathcal{S}$ satisfait φ (ce que l'on note $\mathcal{S} \models \varphi$), on se base sur une exploration partielle ou exhaustive de l'espace d'états du modèle. Les limitations sont encore une fois l' indécidabilité du problème suivant le modèle et les propriétés à vérifier, ainsi que l'explosion combinatoire due à un nombre d'états trop important à analyser.

1.2.1 Modèles temporisés

Nous nous plaçons dans le contexte de la vérification de modèles. Les systèmes temps réel sont un domaine d'application privilégié de ces méthodes. Ils présentent en effet des exigences en termes de qualité et de sûreté qui ne peuvent être apportées que par une approche formelle. La vérification des contraintes temporelles est notamment difficile à réaliser par des tests classiques, puisqu'il faudrait en théorie tester une infinité de séquences temporelles différentes. Une alternative est alors de construire un modèle formel temporisé du système.

La construction d'un modèle formel réalise nécessairement une abstraction du système réel. Le choix du modèle est alors directement lié aux types de propriétés que l'on souhaite vérifier. Ainsi, avec un modèle classique tel que les automates finis on décrit le fonctionnement logique du système : cela permet de vérifier des propriétés temporelles qualitatives, *i.e.* des liens de causalités entre les évènements. Pour vérifier des propriétés temporelles quantitatives, on peut utiliser des modèles temporisés, qui représentent à la fois le fonctionnement logique et les contraintes temporelles quantitatives du système. Bien que l'espace d'états de ces modèles soit en général infini, des méthodes symboliques de model-checking peuvent être utilisées pour vérifier des propriétés temporelles. On distingue plusieurs catégories de modèles temporisés.

Automates temporisés

Les *automates temporisés* sont une extension des automates finis classiques qui intègre des horloges explicites [Alur 94]. Dans une localité, il est possible de rester dans la localité et d'écouler du temps, ou bien de franchir une transition discrète. Les transitions comportent des *gardes* qui spécifient des contraintes sur les horloges : le tir d'une transition n'est possible

que si ses gardes sont vérifiées ; le tir d'une transition peut également réinitialiser certaines horloges. Des invariants peuvent être ajoutés sur les localités [Henzinger 94] : ils restreignent dans ce cas les conditions de séjour dans la localité. Afin de calculer l'espace d'états de ces modèles, tous les états accessibles entre le tir de deux transitions de l'automate peuvent être regroupés à l'intérieur d'une *zone* [Larsen 95]. Une zone peut être représentée par une *matrice de différences bornées*. Deux des principaux outils sur les automates temporisés sont UPPAAL [Larsen 97] et KRONOS [Yovine 97].

Extension temporelles des réseaux de Petri

Les réseaux de Petri sont également étendus avec des informations temporelles. Les deux principales extensions temporelles sont les réseaux de Petri temporisés [Ramchandani 74] et les réseaux de Petri temporels [Merlin 74]. Les premiers considèrent des durées minimales de tir pour les transitions alors que les seconds ajoutent des intervalles temporels de tir aux modèles. Il existe également plusieurs manières d'intégrer le temps dans les réseaux de Petri, en plaçant les informations temporelles soit sur les transitions, soit sur les places, soit sur les arcs.

Les réseaux de Petri T-temporels [Merlin 74, Berthomieu 91] ajoutent un intervalle temporel aux transitions du réseau. En utilisant une sémantique de tir dite « forte », ils sont capables de modéliser l'urgence de certains évènements. Cela permet de représenter des motifs tels que le « chien de garde » très utilisés dans les systèmes temps réel. Pour représenter l'espace d'états de ces modèles on utilise également des méthodes symboliques. La méthode du *graphe des classes d'états* [Berthomieu 91], et celle du *graphe des zones* [Gardey 03], sont les principales méthodes utilisées. Les principaux outils manipulant des réseaux de Petri T-temporels sont TINA [Berthomieu 04], ORIS [Bucci 04b] et ROMEO [Gardey 05b, Lime 09].

Modèles à chronomètres

Les modèles temporisés classiques ne sont pas capables de modéliser des systèmes temps réel avec ordonnancement préemptif. Il est pour cela nécessaire de représenter l'interruption et la reprise de l'exécution d'une tâche. Une solution est de considérer des modèles à chronomètres, dans lesquels la notion d'horloge utilisée dans les modèles temporisée est remplacée par celle de chronomètre. Contrairement à une horloge, un chronomètre peut être interrompu (il conserve sa valeur lors de l'écoulement du temps) puis redémarré. Plusieurs modèles intègrent cette notion.

Les extensions à chronomètres des réseaux de Petri T-temporels se distinguent par leur manière de contrôler les chronomètres : par des priorités [Bucci 04a, Roux 02], par des arcs inhibiteurs [Roux 04] ou encore par des arcs activateurs [Berthomieu 07].

Un modèle plus général est celui des automates hybrides [Alur 95]. Ils forment une généralisation des automates temporisés dans lesquels l'évolution de chaque horloge est contrôlée par des équations différentielles. Les automates à chronomètres sont une sous-classe de ce modèle [Cassez 00] : les chronomètres sont définis en restreignant les dérivés des horloges à prendre comme valeur soit (1), soit (0) . Ces modèles peuvent être utilisés dans l'outil HYTECH [Henzinger 97].

Il faut cependant noter que le gain en expressivité de ces modèles se traduit par l'indécidabilité de la plupart des problèmes intéressants.

1.2.2 Représentation de la concurrence

La concurrence, ou parallélisme, dans un modèle formel correspond aux actions pouvant s'exécuter simultanément. Cette propriété est essentielle pour les systèmes distribuées, dans lesquels de nombreuses tâches peuvent s'exécuter en parallèle. Dans beaucoup de modèles basés sur les systèmes de transitions, notamment les automates, la représentation du parallélisme ne peut être réalisée que par l'entrelacement des actions concurrentes. Dans ces modèles l'état courant correspond à une seule localité : il décrit donc nécessairement l'état global de toutes les tâches concurrentes, et une seule transition de sortie de cet état peut être choisie : donc une seule action peut être exécutée. Au contraire, dans les réseaux de Petri, les multiples jetons peuvent représenter chacun un état dans une tâche différente. Cela permet de représenter explicitement le parallélisme : on parle alors de « parallélisme vrai ». Un réseau d'automates constitué de plusieurs automates en parallèle est également un modèle avec du « parallélisme vrai ».

Toutefois, même dans les modèles conservant le parallélisme, les méthodes utilisées pour analyser ces modèles utilisent en général l'entrelacement des actions concurrentes. Ainsi, les méthodes d'analyse des réseaux de Petri temporels sont basées sur une sémantique séquentielle définie à l'aide de systèmes de transitions temporisés. Or l'entrelacement des actions concurrentes est une des causes qui peut amener au problème de l'explosion combinatoire. Il est donc intéressant de faire appel à des méthodes d'analyse conservant le parallélisme, telles que les réductions des ordres partiels et les dépliages.

1.3 Une approche de vérification paramétrée

Comme nous l'avons présenté, le processus de conception d'un système informatique se compose d'une succession d'étapes, et à chacune de ces étapes des choix de conception sont réalisés, jusqu'à aboutir au système final. Plusieurs modèles du système sont créés pour décrire les différentes vues du système. Ces modèles deviennent de plus en plus raffinés au fur et mesure de la conception. Il est notamment fréquent de les paramétrer afin de modéliser les choix de conception qui ne sont pas encore déterminés. Ces paramètres sont des variables, fixées dans un certain domaine : ils donnent au modèle des degrés de liberté pour décrire le système. Il faut donc voir un modèle paramétré comme une classe de modèles définie par l'ensemble des valeurs possibles des paramètres. Les modèles paramétrés permettent en particulier de définir des éléments de modélisation réutilisables dans différents contextes, par simple changement des valeurs des paramètres.

Les choix effectués pendant la conception doivent être validés par une étape de vérification appropriée. Idéalement, la vérification est réalisée le plus tôt possible, de sorte que si elle échoue il suffit de recommencer l'étape de conception correspondante. Cependant, en pratique, cela n'est pas toujours possible, car certaines propriétés ne peuvent être vérifiées qu'à la fin de la conception, lorsque le système complet est connu. C'est notamment le cas des contraintes temporelles dans les systèmes temps réel : elles ne peuvent en général être vérifiées que sur le système global défini par l'architecture opérationnelle. Dans ces situations, une solution consiste à définir un modèle paramétré du système, dans lequel sont laissées libres les caractéristiques du système qui ne sont pas encore déterminées. Il est alors possible de faire appel à un processus de vérification paramétrée, qui peut être utilisé pour valider les choix de conception selon le résultat obtenu :

- Les propriétés ne sont vérifiées pour aucune valeur des paramètres ; les choix de conception doivent être revus.
- Les propriétés sont vérifiées pour toutes les valeurs des paramètres ; les choix de conception sont validés.
- Les propriétés sont vérifiées pour certaines valeurs des paramètres ; les choix de conception sont validés, à condition d'instancier les caractéristiques paramétrées du système à une valeur correcte. Ces valeurs peuvent alors être définies par des contraintes sur les paramètres.

Nous voyons donc que l'intérêt de la vérification paramétrée est de fournir des résultats plus complets que la simple correction du système. Cela peut servir de guide à la conception en limitant les choix possibles à un ensemble restreint de solutions satisfaisantes.

Par ailleurs, dans une approche modulaire, le système est décomposé en modules à l'aide des contraintes fonctionnelles définissant l'architecture logicielle. Cependant, certaines spécifications ne peuvent pas être partitionnées en un ensemble de spécifications à appliquer à chaque module, mais doivent en théorie être vérifiées sur le système global. C'est notamment le cas des spécifications temporelles. L'analyse du système global, résultant de la composition des modules, pouvant être impossible à réaliser à cause de l'explosion combinatoire, une solution consiste à projeter les contraintes sur les modules du système. Se pose alors la question de la compositionalité : si les contraintes projetées sont vérifiées sur chacun des modules, est-ce que le système global vérifie les spécifications ? Pour s'en assurer, il est possible de réaliser une projection pessimiste et non optimale, mais pouvant rendre difficile la réalisation (l'implémentation) des modules.

Une autre solution est alors de réaliser une projection paramétrée des contraintes sur les modules. Cela permet de réaliser une vérification paramétrée des spécifications sur chacun des modules, ce qui synthétise des contraintes sur les paramètres. La compositionalité peut ensuite être assurée en considérant l'intersection des solutions valides pour les paramètres.

La vérification paramétrée apparaît donc comme un outil permettant de concevoir un système sûr, malgré les incertitudes liées à des spécifications encore incomplètes. L'utilisation d'un modèle paramétré constitue un important gain en expressivité, qui peut par contre engendrer des difficultés pour analyser le modèle.

1.3.1 Notre contribution

Nous souhaitons développer des techniques de vérification paramétrée, pouvant servir notamment à la conception des systèmes temps réel. Nous considérons pour cela le modèle des réseaux de Petri T-temporels et ses extensions à l'ordonnancement. Ces modèles sont un moyen efficace pour modéliser les différents aspects des systèmes temps réel. Ils présentent de plus l'intérêt de modéliser explicitement le parallélisme dans le système.

Nous commençons par paramétrer des réseaux de Petri T-temporels à chronomètres, en remplaçant certaines des valeurs temporelles introduites sur les transitions par des paramètres temporels. Une première approche est alors d'étendre les techniques de model-checking classiques pour définir une méthode de model-checking paramétré. Nous définissons pour cela un graphe des classes d'états paramétrées afin de représenter l'espace d'états d'un modèle paramétré. En utilisant ce graphe des classes d'états paramétrées, nous proposons une méthode de vérification de formules de logique temporelle TCTL (*Temporal Computation Tree Logic*)

également paramétrées. Cette approche de vérification paramétrée est implémentée dans le logiciel Romeo.

Cette approche de vérification paramétrée par model-checking, bien que très puissante, se heurte à la complexité des modèles utilisés. Dans un second temps, nous souhaitons définir une méthode d'analyse préservant le parallélisme décrit par les réseaux de Petri. Cela apparaît particulièrement intéressant pour les modèles paramétrés : en effet, des modules en parallèle évoluant avec des contraintes de temps paramétrées différentes sont difficilement comparables par des entrelacements. Nous proposons donc une méthode de dépliage temporel paramétré, qui se distingue des méthodes existantes en proposant un dépliage plus réduit, ce qui limite d'autant plus l'explosion combinatoire. Ce dépliage est en général infini, et son utilisation pour le model-checking semble difficile. Nous proposons alors de l'utiliser dans des méthodes de vérification alternatives, en présentant une application au problème de la supervision de modèles : le dépliage est guidé par des observations finies et nous synthétisons par cette méthode des contraintes sur les paramètres pour expliquer ces observations.

1.4 Organisation du manuscrit

Le manuscrit est organisé en quatre parties. La première partie, qui inclut cette introduction, présente les définitions des notions utilisées dans le reste du manuscrit.

Le **chapitre 2** présente les notations mathématiques utilisées dans le manuscrit, ainsi que certaines notions de base concernant les systèmes de transitions temporisés et les systèmes de contraintes sur des variables.

Nous introduisons ensuite dans le **chapitre 3** les modèles temporisés que nous étudions, c'est-à-dire les réseaux de Petri temporels et leurs extensions à l'ordonnancement. Nous présentons également les logiques utilisées pour spécifier des propriétés et le calcul de l'espace d'états à l'aide du graphe des classes d'états.

La seconde partie du manuscrit présente notre contribution à l'étude du model-checking paramétré.

Le **chapitre 4** définit le modèle des réseaux de Petri à chronomètres paramétrés que nous utilisons. Nous donnons ensuite une méthode de calcul de l'espace d'états de ce modèle en définissant un graphe des classes d'états paramétrées. Enfin, nous présentons une méthode de model-checking paramétré de formules TCTL paramétrés.

Le **chapitre 5** décrit l'implémentation de notre approche paramétrée dans le logiciel Romeo développé à l'IRCCyN. L'utilisation de cette implémentation est illustrée sur une étude de cas d'un problème d'ordonnancement de tâches.

La troisième partie concerne notre contribution aux méthodes de dépliages des réseaux de Petri temporels.

Dans le **chapitre 6** nous introduisons les notions liées aux dépliages des réseaux de Petri et nous présentons une nouvelle méthode de dépliage de réseaux de Petri temporels paramétrés.

Le **chapitre 7** décrit tout d'abord l'implémentation de cette méthode de dépliage dans Romeo. Dans la suite du chapitre, nous présentons une méthode de résolution d'un problème de la supervision de modèles à l'aide des dépliages temporels paramétrés. Nous appliquons cette méthode sur un exemple paramétré du protocole du bit alterné.

Enfin dans la dernière partie, constituée uniquement du **chapitre 8**, nous concluons ce mémoire en effectuant une synthèse des travaux réalisés, et nous proposons des perspectives de recherche à la suite de ces travaux.

Chapitre 2

Préliminaires

Résumé : *Ce chapitre présente en préliminaire des travaux de thèse les principales notations mathématiques utilisées dans le manuscrit ainsi que certaines notions de base utilisées par la suite. Cela inclut les systèmes de transitions temporisés et les représentations de systèmes de contraintes sur des variables à l'aide de polyèdres ou de matrices de différences bornées.*

Sommaire

2.1 Notations

$\mathbb{N}$, $\mathbb{Q}$ et $\mathbb{R}$ désignent respectivement les ensembles des nombres naturels, rationnels et réels. $\mathbb{R}^+$ (resp. $\mathbb{Q}^+$) est l'ensemble des nombres réels (resp. rationnels) positifs ou nuls, et $\mathbb{R}_*^+$ (resp. $\mathbb{Q}_*^+$) est l'ensemble des nombres réels (resp. rationnels) strictement positifs.

Soient $n \in \mathbb{N}$, $\mathbb{R}^n$ désigne l'espace réel à n dimensions.

Étant donné deux entiers m et n, on notera $[m..n]$ l'ensemble des entiers compris entre m et n inclus.

Soit un ensemble fini E. Nous notons $|E|$ son cardinal. 2^E est l'ensemble des sous-ensembles de E.

Soient A et X deux ensembles. Nous notons A^X l'ensemble des applications de X dans A. Si de plus X est fini et $|X| = n$, alors un élément de A^X est aussi un vecteur de A^n. Soient $f \in A^X$ une application et $Y \subset X$ un sous-ensemble de X, on note $f_{|Y} \in A^Y$ la restriction de f à Y.

Un multi-ensemble sur un ensemble A est une application de A dans $\mathbb{N}$ qui associe à chaque élément de A le nombre d'occurrences de l'élément dans le multi-ensemble. Si A est fini ($|A| = n$) un multi-ensemble est donc également un vecteur de $\mathbb{N}^n$. On peut comparer deux multi-ensembles v, w sur A par la relation $\leq$ (respectivement $<$) définie par : $v \leq w$ (resp. $v < w$) si et seulement si $\forall a \in A,\ v(a) \leq w(a)$ (resp. $v(a) < w(a)$).

Pour un domaine $\mathbb{T}$ ($\mathbb{N}$, $\mathbb{Q}$ ou $\mathbb{R}$) et une dimension $n \in \mathbb{N}$, $\boldsymbol{v} = (v_1, \ldots, v_n)$ est un vecteur de $\mathbb{T}^n$, et $\forall i \in [1..n]$, v_i est la i-ème composante de $\boldsymbol{v}$. $\mathbf{0} \in \mathbb{T}^n$ est le vecteur origine tel que toutes ses composantes sont nulles. Un vecteur $\boldsymbol{v} \in \mathbb{T}^n$ peut être considéré comme une matrice dans $\mathbb{T}^{1\times n}$.

Soient $\boldsymbol{v}, \boldsymbol{w}$ deux vecteurs de $\mathbb{T}^n$, nous noterons $\boldsymbol{v} < \boldsymbol{w}$ (resp. $\boldsymbol{v} \leq \boldsymbol{w}$ et $\boldsymbol{v} = \boldsymbol{w}$) si et seulement si $\forall i \in [1..n],\ v_i < w_i$ (resp. $v_i \leq w_i$ et $v_i = w_i$). Le *produit scalaire* de $\boldsymbol{v}$ et $\boldsymbol{w}$ est $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \sum_{i=1}^n (v_i w_i)$.

Soient $\boldsymbol{v} \in \mathbb{T}^n$ et $\boldsymbol{w} \in \mathbb{T}^m$ deux vecteurs, nous notons $(\boldsymbol{v}|\boldsymbol{w}) \in \mathbb{T}^{n+m}$ le vecteur obtenu par la concaténation de $\boldsymbol{v}$ et $\boldsymbol{w}$, c'est-à-dire tel que $\forall i \in [1..n],\ (\boldsymbol{v}|\boldsymbol{w})(i) = \boldsymbol{v}(i)$ et $\forall i \in [n+1..n+m],\ (\boldsymbol{v}|\boldsymbol{w})(i) = \boldsymbol{w}(i)$.

$\wedge$ désigne l'opérateur logique *et*, $\vee$ l'opérateur logique *ou*, et $\neg$ l'opérateur logique *non*.

Un intervalle I de $\mathbb{R}^+$ est un $\mathbb{Q}^+$-intervalle si et seulement si son extrémité gauche, que nous notons ${}^{\uparrow}I$, appartient à $\mathbb{Q}^+$, et son extrémité droite, que nous notons $I^{\downarrow}$, appartient à $\mathbb{Q}^+ \cup \{\infty\}$. Nous notons $\mathcal{I}(\mathbb{Q}^+)$ l'ensemble des $\mathbb{Q}^+$-intervalles de $\mathbb{R}^+$.

Soit un ensemble fini X de variables, tel que $|X| = n$, prenant leur valeur dans un domaine $\mathbb{T}$. Une *valuation* est une fonction $\nu \in \mathbb{T}^X$. $\forall i \in [1..n]$, $x_i \in X$ est la i-ème variable de X et $\nu(x_i)$ est la valeur de x_i pour la valuation ν. $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n))$ est le vecteur de $\mathbb{T}^n$ correspondant à la valuation ν. En pratique on confondra les valeurs des variables avec le nom de la variable et l'on notera $\nu = (x_1, x_2, \ldots, x_n)$.

Une *expression linéaire* sur X est une expression $\gamma = \sum_{i=1}^n a_i * x_i + b$, où $b \in \mathbb{T}$ et $\forall i \in [1..n],\ x_i \in X,\ a_i \in \mathbb{T}$. Pour une valuation ν des variables X, on peut également considérer la valuation de γ par ν : $[\![\gamma]\!]_\nu = \sum_{i=1}^n a_i * \nu(x_i) + b$.

Une *contrainte linéaire* sur X est une expression $c = \gamma \sim 0$, où $\gamma = \sum_{i=1}^n a_i * x_i + b$ est

une expression linéaire sur X et $\sim \in \{=, <, >, \leq, \geq\}$. On note également $c = \langle \boldsymbol{a}, \boldsymbol{x} \rangle \sim b$, où $\boldsymbol{a} = (a_1, \ldots, a_n)$ et $\boldsymbol{x} = (x_1, \ldots, x_n)$. La valeur de vérité d'une contrainte c pour une valuation ν des variables X est donnée par : $[\![c]\!]_\nu \Leftrightarrow ([\![\gamma]\!]_\nu \sim 0)$.

2.2 Systèmes de transitions temporisés

2.2.1 Définition

Les systèmes de transitions temporisés (Timed Transition System en anglais, noté TTS) constituent un modèle de base pour décrire l'évolution discrète et temporelle d'un système. Ils sont une extension temporisée des systèmes de transitions, classiquement utilisés dans les systèmes à évènements discrets, et peuvent être définis suivant l'idée de [Larsen 95] comme des systèmes de transitions possédant deux types de transitions : des transitions d'actions, décrivant l'évolution discrète, et des transitions de temps, décrivant l'évolution temporelle ou continue.

Les systèmes de transitions temporisés seront utilisés pour exprimer la sémantique des extensions temporisées des réseaux de Petri et des logiques temporelles.

Définition 2.1 (Système de transitions temporisés (TTS)) *Un* système de transitions temporisés *est un quadruplet* $\mathcal{S} = \langle Q, Q_0, \Sigma, \rightarrow \rangle$ *où :*

- Q *est un ensemble d'états ;*
- $Q_0 \subseteq Q$ *est l'ensemble des états initiaux ;*
- Σ *est un alphabet disjoint de* $\mathbb{R}^+$ *;*
- $\rightarrow \subseteq Q \times \{\Sigma \cup \mathbb{R}^+\} \times Q$ *est une relation de transition composée :*
  - *d'une relation de* transition discrète *:* $\xrightarrow{a \in \Sigma} \subseteq Q \times \Sigma \times Q$ *;*
  - *d'une relation de* transition continue *:* $\xrightarrow{\delta \in \mathbb{R}^+} \subseteq Q \times \mathbb{R}^+ \times Q$

Étant donnés deux états $q, q' \in Q$ *et* $e \in \Sigma \cup \mathbb{R}^+$*, si* $(q, e, q') \in \rightarrow$ *on utilise la notation* $q \xrightarrow{e} q'$*. Pour* $\delta \in \mathbb{R}^+$*, si* $q \xrightarrow{\delta} q'$ *on notera également* $q \xrightarrow{\delta} q + \delta$*. Pour* $\delta \in \mathbb{R}+$ *et* $a \in \Sigma$*, si* $q \xrightarrow{\delta} q + \delta \xrightarrow{a} q'$ *on pourra noter* $q \xrightarrow{(\delta, a)} q'$*.*

La relation de transition doit de plus vérifier les propriétés suivantes :

- *déterminisme temporel :* $\forall q, q', q'' \in Q,\ \forall \delta \in \mathbb{R}^+$*, si* $q \xrightarrow{\delta} q'$ *et* $q \xrightarrow{\delta} q''$ *alors* $q' = q''$ *;*
- *additivité du temps :* $\forall q, q'' \in Q,\ \forall \delta_1, \delta_2 \in \mathbb{R}^+$*, si* $q \xrightarrow{\delta_1 + \delta_2} q''$ *alors* $\exists q' \in Q$ *tel que* $q \xrightarrow{\delta_1} q' \xrightarrow{\delta_2} q''$ *;*
- *attente nulle :* $\forall q, q' \in Q,\ q \xrightarrow{0} q'$ *si et seulement si* $q = q'$*.*

2.2.2 Langage

Pour définir le langage d'un système de transitions on considère que tous les états sont accepteurs, et donc n'importe qu'elle séquence de transitions depuis l'état initial forme un mot accepté par le langage.

Définition 2.2 (Séquence de transitions) *Soit* $\mathcal{S} = \langle Q, Q_0, \Sigma, \rightarrow \rangle$ *un système de transitions temporisé, un élément* $\rho \in (\Sigma \times \mathbb{R}^+)^*$ *noté* $\rho = (\delta_1, a_1); \ldots; (\delta_n, a_n)$ *est une* séquence de transitions *de* $\mathcal{S}$ *si* $\exists (q_1, q_2 \ldots q_n, q_{n+1}) \in Q^{n+1}$ *tels que* $q_1 \xrightarrow{(\delta_1, a_1)} q_2 \ldots q_n \xrightarrow{(\delta_n, a_n)} q_{n+1}$*.*
Si de plus $q_1 = q_0$ *alors* ρ *est un* mot *de* $\mathcal{S}$*.*

Le *langage temporisé* d'un système de transitions temporisé est l'ensemble des mots de ce système contenant à la fois des actions discrètes et des actions continues ; le *langage non temporisé* est constitué des mots dans lesquels les actions continues sont abstraites.

Une première possibilité pour comparer deux systèmes de transitions temporisés consiste à étudier l'inclusion ou l'égalité des langages. On pourra comparer à la fois les langages temporisés, et les langages non temporisés.

2.2.3 Quotient

On peut regrouper les états d'un système de transitions $\mathcal{S}$ selon une relation d'équivalence $\mathcal{R}$ en calculant le quotient de $\mathcal{S}$ par $\mathcal{R}$, noté $\mathcal{S}/\mathcal{R}$.

Définition 2.3 (Quotient) *Soient $\mathcal{S} = \langle Q, q_0, \Sigma, \rightarrow \rangle$ un système de transitions et $\mathcal{R} \subseteq Q \times Q$ une relation d'équivalence entre les états de $\mathcal{S}$.*
Le quotient $\mathcal{S}/\mathcal{R}$ de $\mathcal{S}$ par $\mathcal{R}$ est le système de transitions $\langle S, s_0, \Sigma, \rightsquigarrow \rangle$ défini par :

- *S est l'ensemble des classes d'équivalence de Q pour $\mathcal{R}$;*
- *s_0 est la classe d'équivalence de l'état q_0, donc $q_0 \in s_0$;*
- *$\rightsquigarrow$ est définie par : soit $(s, s') \in S \times S$ et $a \in \Sigma$, $s \overset{a}{\rightsquigarrow} s'$ si et seulement si il existe $(q, q') \in Q \times Q$ tels que $q \in s$, $q' \in s'$ et $q \xrightarrow{a} q'$.*

2.2.4 Bisimulation

Une relation de bisimulation permet de définir une équivalence de comportement entre deux systèmes de transitions. Elle assure que toute action de l'un des systèmes peut être *simulée* par l'autre, c'est-à-dire qu'elle a un équivalent dans cet autre système.

Définition 2.4 (Bisimulation) *Soient $\mathcal{S}_1 = \langle Q_1, Q_1^0, \Sigma, \rightarrow_1 \rangle$ et $\mathcal{S}_2 = \langle Q_2, Q_2^0, \Sigma, \rightarrow_2 \rangle$ deux systèmes de transitions étiquetées sur un même alphabet Σ, et $\mathcal{R}$ une relation binaire sur l'ensemble des états des deux systèmes $Q_1 \times Q_2$.*

$\mathcal{R}$ est une relation de bisimulation ssi $\forall (q_1, q_2) \in Q_1 \times Q_2$ t.q. $q_1 \mathcal{R} q_2$, $\forall a \in \Sigma$,

$$\begin{cases} \exists q_1' \in Q_1 \text{ t.q. } q_1 \xrightarrow{a}_1 q_1' \Rightarrow \exists q_2' \in Q_2 \text{ t.q. } q_2 \xrightarrow{a}_2 q_2' \text{ et } q_1' \mathcal{R} q_2', \\ \exists q_2' \in Q_2 \text{ t.q. } q_2 \xrightarrow{a}_2 q_2' \Rightarrow \exists q_1' \in Q_1 \text{ t.q. } q_1 \xrightarrow{a}_1 q_1' \text{ et } q_1' \mathcal{R} q_2' \end{cases}$$

Pour deux systèmes de transitions temporisés $\mathcal{S}_1$ et $\mathcal{S}_2$, on dit que $\mathcal{R}$ est une *relation de bisimulation temporelle* si elle est une relation de bisimulation à la fois pour la relation de transition discrète et pour la relation de transition continue.

Nous remarquons que deux systèmes de transitions bisimilaires ont le même langage. De la même manière deux TTS temporellement bisimilaires ont le même langage temporisé.

2.3 Systèmes de contraintes

Soit un ensemble fini X de variables prenant leur valeur dans un domaine $\mathbb{T}$, tel que $|X| = n$. Un *système de contraintes $\mathcal{C}$* est une combinaison booléenne de contraintes linéaires sur X (à l'aide des opérateurs logiques $\wedge$, $\vee$ et $\neg$).

2.3.1 Polyèdres

On considère l'espace réel en dimension n ($\mathbb{T} = \mathbb{R}$). Une contrainte linéaire $\gamma = \langle \boldsymbol{a}, \boldsymbol{x} \rangle \sim b$, avec deux vecteurs $\boldsymbol{a} \in \mathbb{R}^n$, $\boldsymbol{x} \in X^n$ et un scalaire $b \in \mathbb{R}$, définit :
- un hyperplan affine de $\mathbb{R}^n$ si $\sim \in \{=\}$;
- un demi-espace affine topologiquement fermé si l'inégalité n'est pas stricte : $\sim \in \{\leq, \geq\}$;
- un demi-espace affine topologiquement ouvert sinon : $\sim \in \{<, >\}$.

Définition 2.5 (Polyèdre) *Un* polyèdre convexe *non nécessairement clos est une région de l'espace réel à n dimension définie par l'intersection d'un nombre fini d'espace affines (fermés ou ouverts).*

La relation d'inclusion entre des polyèdres $\mathbb{R}^n$ forme un ordre partiel sur l'ensemble des polyèdres de $\mathbb{R}^n$. L'ensemble vide $\emptyset$ est le plus petit polyèdre alors que $\mathbb{R}^n$ est le plus grand.

Un *polytope* $\mathcal{P}$ est un polyèdre borné, c'est-à-dire qu'il existe $\lambda \in \mathbb{R}^+$ tel que $\mathcal{P} \subseteq \{\boldsymbol{x} \in \mathbb{R}^n \mid \forall i \in [1..n],\ -\lambda \leq x_i \leq \lambda\}$.

Par définition, un polyèdre $\mathcal{P}$ est l'ensemble des solutions d'un système de contraintes en conjonctions, c'est-à-dire la conjonction d'un nombre fini de contraintes linéaires :

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

où pour $i \in \{1, 2, 3\}$ A_i est une matrice de $\mathbb{R}^{m_i \times n}$ et $\boldsymbol{b_i}$ un vecteur de $\mathbb{R}^{m_i}$ (où m_1 est le nombre d'égalités, m_2 le nombre d'inégalités larges et m_3 le nombre d'inégalités strictes).

Une autre représentation possible d'un polyèdre est sous la forme d'un système générateur (union d'un polytope et d'un cône polyédrique [Schrijver 86]).

Les deux représentations sous forme de système de contraintes ou de système générateur sont équivalentes et admettent des formes minimales et une forme canonique unique à un choix de base de sous-espace vectoriel près [Komei 02]. Dans ce manuscrit nous utiliserons uniquement la représentation sous forme de systèmes de contraintes plus adaptée notamment pour l'expression des domaines de tir des classes d'états.

Étant donné un polyèdre $\mathcal{P}$ défini sur un ensemble de variables X, et un sous-ensemble $Y \subset X$ de variables, on définit la projection de $\mathcal{P}$ sur Y notée $\mathcal{P}_{|Y}$ par :

$$\mathcal{P}_{|Y} = \{\nu \in \mathbb{R}^Y \mid \exists \nu' \in \mathbb{R}^X \text{ t.q. } \forall x \in Y,\ \nu(x) = \nu'(x)\}$$

2.3.2 Matrices de différences bornées

Un polyèdre, solution d'un système de contraintes $A\boldsymbol{x} \leq \boldsymbol{b}$ dans lequel la matrice $A \subseteq \mathbb{R}^{m \times n}$ contient au plus un coefficient 1 sur chaque ligne, au plus un -1, et sinon que des 0, admet une représentation simplifiée sous la forme d'une matrice unique appelée *matrice des différences bornées* (*Difference Bound Matrix*, notée DBM) [Berthomieu 83, Dill 89].

Les inéquations de ce système sont en effet de la forme $x_i - x_j \leq c$, $x_i \leq c$ ou $-x_i \leq -c$, pour $i, j \in [1..n]$. On définit alors une variable x_0 supplémentaire telle qu'en tout point de l'espace $x_0 = 0$. Elle permet d'écrire toutes les contraintes sous la forme $x_i - x_j \leq c$ avec $i, j \in [0..n]$. Ce système se représente alors par une matrice carrée $\mathcal{D}$ de taille $n + 1$ dont les coefficients sont soit un réel, soit fixé à ∞. Le coefficient d_{ij} de la i-ème ligne et j-ème colonne de la matrice $\mathcal{D}$ est égal à c si la contrainte $x_i - x_j \leq c$ existe et ∞ sinon.

2.3.3 Complexité

Les polyèdres pouvant être écrits sous la forme d'une DBM présentent l'intérêt d'être plus efficaces à manipuler. En effet, soit n la dimension de l'espace, pour les DBM les algorithmes de test d'inclusion, d'intersection et d'égalité ont des complexités quadratiques en n ($O(n^2)$). La mise sous forme canonique et le test du vide ont une complexité en $O(n^3)$.

Au contraire pour un polyèdre général en dimension n possédant m sommets, la complexité au pire cas de la mise sous forme canonique est polynomiale en n et en m [Komei 02].

Chapitre 3

Modèles temporisés

Résumé : *Plusieurs modèles peuvent être utilisés pour modéliser des systèmes temps réel. Une première classe regroupe notamment les réseaux de Petri temporels et les automates temporisés. Ces modèles peuvent être étendus avec des chronomètres afin de représenter des systèmes préemptifs. On présente dans ce chapitre les différents modèles utilisés dans le manuscrit, on se focalisera donc sur les extensions des réseaux de Petri. On introduit les notions afférentes à ces modèles : les logiques utilisées pour spécifier des propriétés et les méthodes de calcul de l'espace d'états.*

Sommaire

3.1 Réseaux de Petri temporels

Les systèmes de transitions temporisés sont capables de décrire les comportements des systèmes temps réel ; ils ne sont cependant pas efficaces pour manipuler l'ensemble des comportements de ces systèmes. On préfère pour cela utiliser des modèles de plus haut niveau, tels que les extensions temporelles des réseaux de Petri [Merlin 74, Ramchandani 74] ou les automates temporisés [Alur 94]. Les systèmes de transitions temporisés sont alors utilisés pour décrire la sémantique séquentielle de ces modèles. Nous traiterons principalement dans ce manuscrit des extensions des réseaux de Petri.

Parmi les extensions des réseaux de Petri plusieurs intègrent des informations temporelles aux modèles. Les deux principales sont les réseaux de Petri temporisés [Ramchandani 74] et les réseaux de Petri temporels [Merlin 74]. Les premiers considèrent des durées minimales de tir pour les transitions alors que les seconds ajoutent des intervalles temporels de tir aux modèles. Il existe également plusieurs manières d'intégrer le temps dans les réseaux de Petri, en plaçant les informations temporelles soit sur les transitions, soit sur les places, soit sur les arcs. Dans le cas des réseaux de Petri temporels les modèles correspondants sont respectivement les réseaux de Petri T-temporels [Merlin 74, Berthomieu 91], P-temporels [Khansa 96, Khansa 97] et A-temporels [de Frutos Escrig 00, Abdulla 01, Hanisch 93]. Pour chacun de ces modèles, le modèle de réseau Petri temporisé qui lui est associé est moins expressif [Pezze 99].

Nous nous intéressons plus précisément aux réseaux de Petri T-temporels, ou *time Petri nets* (TPN) en anglais, dans lesquels un intervalle temporel est associé à chaque transition. Plusieurs sémantiques de temps sont alors possibles, en temps dense ou en temps discret. En temps dense les deux principales sémantiques sont la sémantique dite « faible » qui autorise une transition à ne pas être franchie même si l'on atteint la borne maximale de l'intervalle, et au contraire une sémantique dite « forte » qui force le tir des transitions qui atteignent leur borne maximale. Dans le cadre des systèmes temps réel cette dernière sémantique permet de modéliser l'urgence de certains évènements ; elle est donc plus appropriée. Enfin, il existe également des sémantiques mono-serveur ou multi-serveurs, lorsque que la multi-sensibilisation des transitions est gérée avec une seule horloge ou plusieurs [Boyer 01, Berthomieu 01]. Pour plus de détails, le lecteur intéressé pourra trouver dans [Boyer 08] une comparaison de l'expressivité des différents modèles de réseaux de Petri temporels, en sémantique faible ou forte.

3.1.1 Syntaxe

Un réseau de Petri est un ensemble de places et de transitions reliées entre elles par des arcs, ce que l'on peut représenter graphiquement par un graphe biparti orienté. Le marquage du réseau est représenté par des jetons disposés à l'intérieur des places. Les arcs peuvent être valués par un poids qui indique alors le nombre de jetons consommés ou produits par une transition lorsqu'il est supérieur à 1.

Les réseaux de Petri T-temporels (TPN) étendent les réseaux de Petri en ajoutant à chaque transition t un intervalle temporel de la forme $I_s(t) = [\mathsf{eft}(t), \mathsf{lft}(t)]$ ($\mathsf{lft}(t)$ pouvant aussi être infini). $\mathsf{eft}(t)$ est la *date de tir au plus tôt* de la transition t, alors que $\mathsf{lft}(t)$ est sa *date de tir au plus tard* ; si $\mathsf{lft}(t)$ est infini la transition ne possède pas de date de tir au plus tard.

Un exemple de TPN est présenté sur la figure 3.1.

Définition 3.1 (Réseau de Petri T-temporel (TPN)) *Un réseau de Petri T-temporel est un 6-uplet* $\mathcal{N} = \langle P, T, {}^\bullet(.), (.)^\bullet, M_0, I_s \rangle$*, où :*

- $P = \{p_1, p_2, \ldots, p_m\}$ *est un ensemble non vide de* places*,*
- $T = \{t_1, t_2, \ldots, t_n\}$ *est un ensemble non vide de* transitions*, tel que* $P \cap T = \emptyset$*,*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *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,*
- $I_s \in \mathcal{I}(\mathbb{Q}^+)^T$ *est la fonction associant un* intervalle temporel de tir *à chaque transition.*

Un *marquage* M du réseau est un élément de $\mathbb{N}^P$, c'est-à-dire un multi-ensemble sur l'ensemble des places 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 $M \geq {}^\bullet t$, (*i.e.* si le nombre de jetons de M dans chaque place d'entrée en amont de t est plus grand ou égal à la valeur de l'arc entre cette place et la transition). Nous notons cela $t \in \mathsf{enabled}\,(M)$.

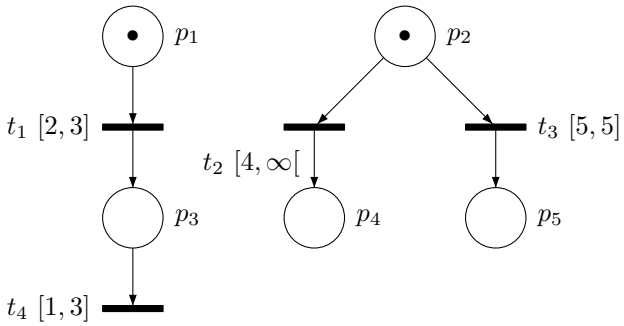


FIG. 3.1: Réseau de Petri T-temporel

3.1.2 Sémantique

Nous adoptons une sémantique forte de type mono-serveur. Cela signifie qu'une horloge est associée à chaque transition sensibilisée et progresse avec le temps. Le tir d'une transition sensibilisée est possible si son horloge a atteint la borne minimale de son intervalle de tir. Lorsqu'une horloge atteint la borne maximale de l'intervalle de tir de sa transition le temps ne peut plus progresser et le tir d'une transition est nécessaire. Le tir d'une transition consomme des jetons dans les places d'entré en amont et produit des jetons dans les places de sortie en aval. Certaines transitions sont alors désensibilisées et d'autres nouvellement sensibilisées. Ces dernières initialisent leur horloge à zéro.

Pour définir les états d'un TPN, nous choisissons de représenter l'horloge d'une transition t par un intervalle temporel $I(t)$. Il sera initialisé à $I_s(t)$, et ses bornes décroîtrons en écoulant du temps.

Définition 3.2 (État d'un TPN) *Un* état *d'un réseau de Petri T-temporel est une paire* $q = (M, I)$ *dans laquelle* M *est un marquage et* $I \in (\mathcal{I}(\mathbb{Q}))^T$ *est une fonction qui associe un intervalle temporel à chaque transition sensibilisée par* M*.*

Une transition t est dite *tirable* dans un état $q = (M, I)$ si elle est sensibilisée depuis au moins $\mathsf{eft}\,(t)$ unités de temps, c'est-à-dire si ${}^\uparrow I(t) = 0$.

Deux transitions t_1, t_2 sont dites en *conflit structurel* si elles partagent une place d'entrée commune (*i.e.* ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$). Elles sont en *conflit effectif* si de plus il existe un état q pour lequel t_1 et t_2 sont tirables.

Une transition t_k est dite *nouvellement sensibilisée* par le tir d'une transition t_i à partir d'un marquage M, ce que nous notons $\uparrow \mathsf{enabled}\,(t_k, M, t_i)$, si cette transition est sensibilisée par le nouveau marquage $M - {}^\bullet t_i + t_i^\bullet$ mais ne l'était pas par $M - {}^\bullet t_i$, où M est le marquage du réseau avant le tir de t_i. Formellement :

$$\uparrow \mathsf{enabled}\,(t_k, M, t_i) = ({}^\bullet t_k \leq M - {}^\bullet t_i + t_i^\bullet) \wedge ((t_k = t_i) \vee ({}^\bullet t_k > M - {}^\bullet t_i))$$

Par extension, nous noterons $\uparrow \mathsf{enabled}\,(M, t_i)$ l'ensemble des transitions nouvellement sensibilisées par le tir d'une transition t_i à partir d'un marquage M.

La sémantique d'un TPN est définie par un système de transitions temporisé, étiqueté avec les transitions du réseau. Les *transitions continues* permettent d'écouler le temps alors que les *transitions discrètes* correspondent au tir d'une transition du réseau de Petri.

Définition 3.3 (Sémantique d'un TPN) *La sémantique d'un réseau de Petri T-temporel en temps dense $\mathcal{N} = \langle P, T, {}^\bullet(.), (.)^\bullet, M_0, I_s\rangle$ est définie par le système de transitions temporisé $\mathcal{S}_\mathcal{N} = \langle Q, q_0, T, \rightarrow\rangle$ tel que :*

- $Q \subseteq \mathbb{N}^P \times (\mathcal{I}(\mathbb{Q}))^T$,
- $q_0 = (M_0, I_s)$,
- $\rightarrow \in Q \times (T \cup \mathbb{R}^+) \times Q$ *est la relation de transition qui inclut la relation de transition continue et la relation de transition discrète.*
  La relation de transition continue est définie $\forall d \in \mathbb{R}^+$ par :

$$(M, I) \xrightarrow{d} (M, I') \text{ ssi } \forall t_i \in T, \begin{cases} {}^\uparrow I'(t_i) = \max(0, {}^\uparrow I(t_i) - d), \\ I'(t_i)^\downarrow = I(t_i)^\downarrow - d, \\ M \geq {}^\bullet t_i \Rightarrow I'(t_i)^\downarrow \geq 0 \end{cases}$$

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

$$(M, I) \xrightarrow{t_i} (M', I') \text{ ssi } \begin{cases} t_i \in \mathsf{enabled}\,(M)\,, \\ M' = M - {}^\bullet t_i + t_i^\bullet, \\ {}^\uparrow I(t_i) = 0, \\ \forall t_k \in T, I'(t_k) = \begin{cases} I_s(t_k) \text{ si } \uparrow \mathsf{enabled}\,(t_k, M, t_i)\,, \\ I(t_k) \text{ sinon.} \end{cases} \end{cases}$$

Une *exécution* ρ dans $\mathcal{S}_\mathcal{N}$ est une séquence de transitions, finie ou infinie, alternant transitions continues et transitions discrètes de la forme [1] :

$$\rho = q_1 \xrightarrow{(d_1,t_1)} q_2 \xrightarrow{(d_2,t_2)} q_3 \cdots$$

On note *first*(ρ) le premier état de ρ ; une exécution est dite *initiale* si *first*$(\rho) = q_0$; les exécutions de $\mathcal{N}$ sont les exécutions initiales de $\mathcal{S}_\mathcal{N}$. Pour un état q, l'ensemble des exécutions maximales à partir de q est noté $\pi(q)$; l'ensemble des exécutions maximales de $\mathcal{N}$ est alors $\pi(q_0)$.

[1] Bien que la sémantique d'un TPN admette les séquences infinies de transitions continues, les exécutions d'un TPN se restreignent à agglomérer les transitions continues en nombre fini seulement, *i.e.* $\forall i, d_i \in \mathbb{R}^+$.

Exemple 3.1 *Dans le TPN présenté sur la figure 3.1, l'état initial q_0 est constitué par le marquage $\{p_1, p_2\}$ pour lequel les transitions t_1, t_2 et t_3 sont sensibilisées, et dont les intervalles temporels sont respectivement $[2,3]$,$[4,\infty[$ et $[5,5]$. On construit une exécution ρ :*

- *initialement : $\rho = q_0$;*
- *dans l'état initial seule t_1 est tirable ; on choisit d'écouler 3 unités de temps :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3$, $I(t_1) = [0,0]$, $I(t_2) = [1,\infty[$, $I(t_3) = [2,2]$;
- *t_1 est alors urgente : on ne peut plus écouler de temps ; on tire t_1 :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1$, $I(t_2) = [1,\infty[$, $I(t_3) = [2,2]$, $I(t_4) = [1,3]$;
- *t_2, t_3 et t_4 ne sont toujours pas tirables ; on écoule 2 nouvelles unités de temps :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1 \xrightarrow{2} q_1 + 2$, $I(t_2) = [0,\infty[$, $I(t_3) = [0,0]$, $I(t_4) = [0,1]$;
- *les trois transitions sont maintenant tirables, t_3 est même urgente ; on peut alors choisir de façon non déterministe la transition à tirer :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1 \xrightarrow{2} q_1 + 2 \xrightarrow{t_2} q_2$, $I(t_4) = [0,1]$;
- *on écoule 1 nouvelle unité de temps :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1 \xrightarrow{2} q_1 + 2 \xrightarrow{t_2} q_2 \xrightarrow{1} q_2 + 1$, $I(t_4) = [0,0]$;
- *et finalement on tire t_4 :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1 \xrightarrow{2} q_1 + 2 \xrightarrow{t_2} q_2 \xrightarrow{1} q_2 + 1 \xrightarrow{t_4} q_3$.

Le marquage final atteint après l'exécution ρ est $\{p_4\}$; plus aucune transition n'est sensibilisée.

Soit une exécution $\rho = q_1 \xrightarrow{(d_1,t_1)} q_2 \xrightarrow{(d_2,t_2)} \dots$. Pour $i \geq 1$, q_i est le $i^{\text{ème}}$ état atteint dans l'exécution ρ ; le temps absolu écoulé à l'état q_i (depuis l'état initial) est défini par $\mathsf{time}(\rho, q_i) = d_1 + d_2 + \dots + d_{i-1}$; le temps total écoulé dans l'exécution ρ est $\mathsf{time}(\rho) = \sum_{i \geq 1} d_i$.

Un TPN sera appelé *zénon* s'il possède des exécutions infinies ρ telles que $\mathsf{time}(\rho)$ est fini. Un exemple de TPN zénon est présenté sur la figure 3.2. En pratique, on exclura en général les cas zénon, ce qui se vérifie par une simple analyse structurelle en vérifiant que le réseau ne possède pas de cycles dont la somme des bornes minimales des intervalles temporels est nulle.

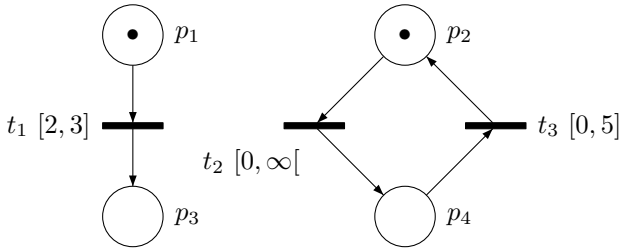


FIG. 3.2: Réseau de Petri T-temporel zénon

3.1.3 Quelques résultats

Étant donné un réseau de Petri T-temporel $\mathcal{N}$ et sa sémantique $\mathcal{S}_{\mathcal{N}} = \langle Q, q_0, \to \rangle$, les principaux problèmes qui peuvent être étudiés sont :

- l'*accessibilité de marquage* : étant donné un marquage M, « $\exists (M', I') \in Q,\ M = M'$ » ;
- la *bornitude* : « $\exists b \in \mathbb{N},\ \forall (M, I) \in Q,\ \forall p \in P,\ M(p) \leq b$ » ;
- la *k-bornitude* : étant donné $k \in \mathbb{N}$ « $\forall (M, I) \in Q,\ \forall p \in P,\ M(p) \leq k$ » ;

– l'*accessibilité d'état* : étant donné un état q, « $q \in Q$ » ;
– la *vivacité* : « $\forall t \in T,\ \forall q \in Q,\ \exists \sigma \in T^*, q' \in Q,\ q \xrightarrow{\sigma.t} q'$ ».

L'accessibilité de marquage est prouvée indécidable dans [Jones 77], ce qui implique que la bornitude, l'accessibilité d'état et la vivacité sont également des problèmes indécidables. Par contre, la k-bornitude est un problème décidable dans les réseaux de Petri T-temporels, en utilisant par exemple la méthode du graphe des classes d'états présentée par la suite [Berthomieu 83, Berthomieu 91].

Pour les réseaux T-temporels bornés, l'accessibilité de marquage, l'accessibilité d'état et la vivacité deviennent des problèmes décidables.

Dans le cas de la 1-bornitude le réseau sera dit *sauf*.

3.2 Réseaux de Petri à chronomètres

L'introduction du temps dans les réseaux de Petri permet de modéliser et de vérifier certains systèmes temps réel mais cela n'est pas suffisant pour de nombreuses applications. En effet le temps décrit l'évolution du système de manière globale et ne peut pas être suspendu. Or dans beaucoup d'applications l'évolution temporelle est au contraire locale (sur un processeur par exemple), et le système évolue à plusieurs vitesses dans des parties différentes. Certaines tâches peuvent notamment être suspendues puis reprises. Cela est en particulier nécessaire pour modéliser des applications d'ordonnancement avec préemption.

Plusieurs modèles ont été proposés pour inclure la notion d'actions pouvant être suspendues et reprises. Cela nécessite d'étendre la notion traditionnelle d'horloge par celle de *chronomètres*. Les automates hybrides [Alur 95] sont le modèle le plus général et le plus répandu. Ils forment une généralisation des automates temporisés dans lesquels l'évolution de chaque horloge est contrôlée par des équations différentielles. Les automates à chronomètres se définissent comme une sous-classe des automates hybrides linéaires dans laquelle les dérivés des horloges ne peuvent prendre comme valeur que (1) ou (0) [Cassez 00].

Dans les réseaux de Petri plusieurs extensions des TPN proposent ces fonctionnalités. Elles se distinguent par leur manière de contrôler les chronomètres, soit par des priorités [Bucci 04a, Roux 02], soit par des arcs inhibiteurs [Roux 04] ou encore par des arcs activateurs [Berthomieu 07]. Tous ces modèles appartiennent à la classe des réseaux de Petri T-temporels à chronomètres [Berthomieu 07] (*Petri nets with stopwatches* en anglais) que nous noterons SwPN.

3.2.1 Syntaxe

Nous choisissons de présenter les réseaux de Petri à chronomètres en utilisant des arcs inhibiteurs. Un *arc inhibiteur* relie une place à une transition ; le chronomètre associé à la transition sera arrêté si un jeton est présent dans une des places reliées à la transition par un arc inhibiteur. Ce modèle est appelé réseau de Petri temporel avec arcs inhibiteurs ; nous utiliserons l'abréviation ITPN (pour *Inhibitor time Petri net*). Un exemple d'ITPN est présenté sur la figure 3.3. Dans cet exemple un arc inhibiteur relie la place p_1 à la transition t_2 ; il est représenté par un arc terminé par un cercle. Ce modèle offre ainsi l'avantage d'une représentation syntaxique et graphique simple de l'inhibition d' actions. Les développements

qui utilisent ce modèle peuvent être généralisés aux autres modèles de réseaux de Petri à chronomètres de façon immédiate.

Définition 3.4 (Réseau de Petri temporel avec arcs inhibiteurs (ITPN)) *Un réseau de Petri temporel avec arcs inhibiteurs est un 7-uplet* $\mathcal{N} = \langle P, T, {}^\bullet(.), (.)^\bullet, {}^\circ(.), M_0, I_s \rangle$ *tel que :*

- $\langle P, T, {}^\bullet(.), (.)^\bullet, M_0, I_s \rangle$ *est un TPN,*
- ${}^\circ(.) \in (\mathbb{N}^P)^T$ *est la* fonction d'inhibition*,*

Une transition t est dite *inhibée* par un marquage M si une place connectée à t par un arc inhibiteur possède au moins autant de jetons que le poids de l'arc inhibiteur :

$$\exists p \in P,\ 0 < {}^\circ t(p) \leq M(p)$$

Nous notons cela $t \in \mathsf{inhibited}(M)$.

Une transition est alors dite *active* dans le marquage M si elle est sensibilisée mais pas inhibée par M.

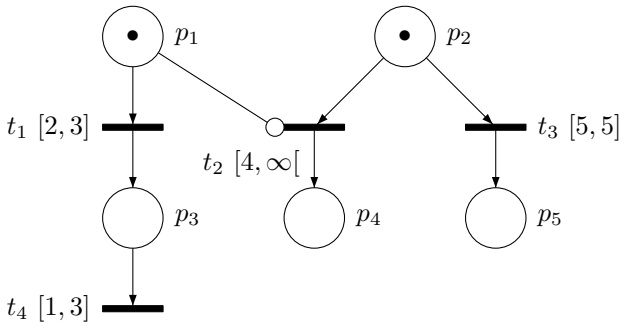


FIG. 3.3: Réseau de Petri temporel avec arcs inhibiteurs

3.2.2 Sémantique

Par rapport aux TPN, l'introduction des arcs inhibiteurs modifie l'évolution des horloges pour les transitions inhibées : le temps est suspendu ; ainsi l'horloge (ou le chronomètre en l'occurrence) conserve sa valeur. Par ailleurs on ne pourra franchir une transition sensibilisée que si elle n'est pas inhibée par le marquage courant, c'est-à-dire active.

Définition 3.5 (Sémantique d'un ITPN) *La sémantique d'un réseau de Petri temporel avec arcs inhibiteurs* $\mathcal{N} = \langle P, T, {}^\bullet(.), (.)^\bullet, {}^\circ(.), M_0, I_s \rangle$ *est définie par le système de transitions temporisé* $\mathcal{S}_\mathcal{N} = \langle Q, q_0, T, \rightarrow \rangle$ *tel que :*

- $Q \subseteq \mathbb{N}^P \times (\mathcal{I}(\mathbb{Q}))^T$*,*
- $q_0 = (M_0, I_s)$*,*
- $\rightarrow \in Q \times (T \cup \mathbb{R}^+) \times Q$ *est la relation de transition qui inclut la relation de transition continue et la relation de transition discrète.*

La relation de transition continue est définie $\forall d \in \mathbb{R}^+$ *par :*

$$(M,I) \xrightarrow{d} (M,I') \text{ ssi } \forall t_i \in T,$$
$$\begin{cases} I'(t_i) = \begin{cases} I(t_i), \text{ si } t_i \in \mathsf{enabled}(M) \text{ et } t_i \in \mathsf{inhibited}(M), \\ {}^{\uparrow}I'(t_i) = \max(0, {}^{\uparrow}I(t_i) - d), \text{ et } I'(t_i)^{\downarrow} = I(t_i)^{\downarrow} - d, \text{ sinon,} \end{cases} \\ M \geq {}^{\bullet}t_i \Rightarrow I'(t_i)^{\downarrow} \geq 0 \end{cases}$$

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

$$(M,I) \xrightarrow{t_i} (M',I') \text{ ssi } \begin{cases} t_i \in \mathsf{enabled}(M) \text{ et } t_i \notin \mathsf{inhibited}(M), \\ M' = M - {}^{\bullet}t_i + t_i^{\bullet}, \\ {}^{\uparrow}I(t_i) = 0, \\ \forall t_k \in T, I'(t_k) = \begin{cases} I_s(t_k) \text{ si } \uparrow\mathsf{enabled}(t_k, M, t_i), \\ I(t_k) \text{ sinon.} \end{cases} \end{cases}$$

Exemple 3.2 *Dans l'ITPN présenté sur la figure 3.3, le marquage initial sensibilise* t_1, t_2 *et* t_3 *et inhibe* t_2*. On construit une exécution* ρ *:*

- *initialement :* $\rho = q_0$, $I(t_1) = [2,3]$, $I(t_2) = [4,\infty[$ *et* $I(t_3) = [5,5]$ *;*
- *on choisit d'écouler 3 unités de temps ; les intervalles temporels de* t_1 *et* t_3 *décroissent, mais pas celui de* t_2 *puisqu'elle est inhibée :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3$, $I(t_1) = [0,0]$, $I(t_2) = [4,\infty[$ *et* $I(t_3) = [2,2]$ *;*
- *on tire* t_1 *qui est urgente :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1$, $I(t_2) = [4,\infty[$, $I(t_3) = [2,2]$ *et* $I(t_4) = [1,3]$ *;*
- t_2 *est maintenant désinhibée ; on écoule 2 nouvelles unités de temps :*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1 \xrightarrow{2} q_1 + 2$, $I(t_2) = [2,\infty[$, $I(t_3) = [0,0]$ *et* $I(t_4) = [0,1]$ *;*
- t_2 *n'est pas encore tirable mais* t_3 *est urgente ; on a le choix de tirer* t_3 *ou* t_4 *:*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1 \xrightarrow{2} q_1 + 2 \xrightarrow{t_4} q_2$, $I(t_2) = [2,\infty[$ *et* $I(t_3) = [0,0]$ *;*
- *on est maintenant obliger de tirer* t_3 *:*
  $\rho = q_0 \xrightarrow{3} q_0 + 3 \xrightarrow{t_1} q_1 \xrightarrow{2} q_1 + 2 \xrightarrow{t_4} q_2 \xrightarrow{t_3} q_3$.

Le marquage final atteint après l'exécution ρ *est cette fois* $\{p_5\}$ *; plus aucune transition n'est sensibilisée.*

3.2.3 Quelques résultats

Les problèmes d'accessibilité de marquage et d'état, de bornitude et de vivacité étant indécidables dans les réseaux de Petri T-temporels [Jones 77] ils le sont également dans le cas des réseaux de Petri à chronomètres en temps dense. Il est à noter que ces problèmes sont décidables si une sémantique en temps discret est adoptée [Magnin 07].

Mais dans les réseaux de Petri à chronomètres en temps dense l'accessibilité de marquage et d'état, la vivacité et la k-bornitude sont indécidables même dans les cas bornés [Lime 04a, Berthomieu 07].

3.3 Expression de propriétés

Le choix et la conception d'un modèle formel d'un système vont de pair avec la spécification des propriétés qui devront être vérifiées sur le modèle. Les propriétés doivent pour cela être spécifiées dans un formalisme adapté au modèle considéré. Une propriété de base dans les

modèles formels est l'accessibilité d'état. Une première approche pour spécifier des propriétés plus évoluées, consiste à enrichir le modèle afin de réduire la vérification de la propriété à l'accessibilité d'un état du modèle enrichi ; une seconde approche consiste à utiliser des logiques et des algorithmes dédiées.

3.3.1 Observateurs

Dans les réseaux de Petri temporels (et les réseaux de Petri à chronomètres) un observateur est un réseau de Petri temporel ajouté au modèle initial de *manière non intrusive* (c'est-à-dire en ne modifiant pas le comportement du modèle initial) et qui permet de déterminer la valeur de vérité d'une propriété [Toussaint 97]. Pour cela, cette valeur de vérité peut être donnée par l'accessibilité d'un marquage ou par l'occurrence du tir d'une transition dans l'espace d'états du modèle enrichi par l'observateur.

L'intérêt de cette approche est de transformer la vérification d'une propriété en un problème d'accessibilité, facilement déterminable par les méthodes d'exploration de l'espace d'états. Elle présente cependant plusieurs inconvénients. Premièrement, vis-à-vis de la facilité d'utilisation : la construction d'observateurs peut être une opération fastidieuse, d'autant plus qu'il n'existe pas de méthode de génération automatique d'observateurs. Deuxièmement, vis-à-vis de la performance. D'une part l'utilisation d'observateurs complexifie le modèle initial (l'observateur étant parfois de taille aussi importante que le modèle à vérifier) ce qui peut amener rapidement au problème de l'explosion combinatoire. D'autre part un calcul de l'espace d'états est nécessaire pour chaque propriété à vérifier.

Exemple 3.3 *Un exemple d'observateur pour le réseaux de Petri de la figure 3.3 est présenté sur la figure 3.4 (l'observateur est dessiné en pointillé et ajouté au réseau initial). Il permet de vérifier que la propriété suivante « t_3 est toujours tirée strictement moins de 4 unités de temps après t_1 » est vrai. Cela se traduit par l'absence d'une occurrence du tir de t_{obs} dans l'espace d'états. En effet, on observe le tir de t_1 dans p_6 et celui de t_3 dans p_7 ; 4 unités de temps après t_1, si t_{obs} est tirée cela signifie que t_3 n'a pas été tirée strictement avant cette date, et donc que la propriété est fausse.*

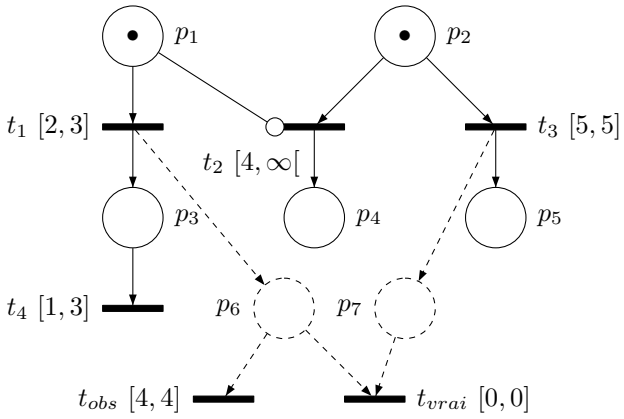


FIG. 3.4: Exemple d'observateur pour un réseau de Petri temporel avec arcs inhibiteurs

3.3.2 Logiques temporelles

Les logiques temporelles fournissent un formalisme spécifique pour exprimer des propriétés.

3.3.2.1 Logiques temporelles non quantitatives (LTL, CTL, CTL*)

Définition 3.6 (CTL* [Emerson 86]) *Soit AP un ensemble de propositions atomiques, les formules CTL* sont définies inductivement par l'ensemble des* formules d'états φ_s*, exprimées par des* formules de chemins φ_p*, selon la grammaire suivante où* $p \in AP$ *:*

$$\varphi_s := \textbf{true} \mid \textbf{false} \mid p \mid \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \varphi_s \vee \varphi_s \mid \forall\varphi_p \mid \exists\varphi_p$$

$$\varphi_p := \varphi_s \mid \neg\varphi_p \mid \varphi_p \wedge \varphi_p \mid \varphi_p \vee \varphi_p \mid \bigcirc \varphi_p \mid \varphi_p \,\mathcal{U}\, \varphi_p \mid \Diamond\varphi_p \mid \Box\varphi_p$$

On interprète les formules sur un système de transitions. Les deux quantificateurs de chemins expriment la sémantique de la formule à un branchement :

- $\forall$: « pour tous », (noté aussi A pour *Always*), la formule est vraie si elle est vérifiée pour tous les successeurs,
- $\exists$: « il existe », (noté aussi E pour *Exists*), la formule est vraie si elle est vérifiée pour au moins un successeur.

Les quatre opérateurs temporels expriment la sémantique de la formule sur une trace :

- $\bigcirc$: « suivant », (noté aussi X pour *neXt*), la formule est vraie si elle est vérifiée dans le successeur immédiat sur le chemin,
- $\varphi \,\mathcal{U}\, \psi$: « jusqu'à », la formule est vraie si dans tous les états futurs du chemin φ est vérifiée jusqu'à un état q_i, et ensuite ψ est vérifiée en q_{i+1},
- $\Diamond$: « finalement », (noté aussi F pour *Finally*), la formule est vraie si elle est vérifiée dans un état futur du chemin ; il peut être défini par $\Diamond\varphi_p = \textbf{true}\,\mathcal{U}\varphi_p$,
- $\Box$: « globalement », (noté aussi G pour *Globally*), la formule est vraie si elle est vérifiée dans tous les états suivants du chemin ; il peut être défini par $\Box\varphi_p = \neg\Diamond\neg\varphi_p$.

Plusieurs sous-classes de CTL* ont été étudiées, notamment :

CTL (*Computational Tree Logic*) [Emerson 82] est le fragment de CTL* restreint de sorte que chaque quantificateurs de chemins est associé à un opérateur temporel. Il est ainsi constitué des combinaisons booléennes de :

$$\forall \bigcirc \varphi,\ \forall\Box\varphi,\ \forall\Diamond\varphi,\ \forall\varphi \,\mathcal{U}\, \psi$$

$$\exists \bigcirc \varphi,\ \exists\Box\varphi,\ \exists\Diamond\varphi,\ \exists\varphi \,\mathcal{U}\, \psi$$

LTL (*Linear Temporal Logic*) [Pnueli 77] est défini uniquement avec les opérateurs temporels. Elle constitue le fragment de CTL* des formules du type $\forall\varphi$ où φ ne comporte plus de quantificateurs de chemins.

Ces deux sous-classes de CTL* ne sont pas comparables. Il existe des formules CTL ne pouvant pas être exprimées en LTL et réciproquement [Emerson 86].

3.3.2.2 Logiques temporelles temporisées (TCTL)

La logique CTL* permet d'exprimer des propriétés sur l'ordonnancement temporel des évènements, mais elle ne permet pas d'exprimer des propriétés temporelles quantitatives. Ainsi la propriété exprimée par un observateur dans l'exemple 3.3 : « t_3 est toujours tirée strictement moins de 4 unités de temps après t_1 » ne peut pas être exprimée en CTL*.

Pour cela des logiques temporelles quantitatives ont été introduites. La logique TCTL (*Temporal Computational Tree Logic*) [Alur 90] est une extension de CTL qui intègre des contraintes temporelles quantitatives. Nous considérons une logique TCTL dans laquelle un intervalle temporel est ajouté sur les opérateurs temporels afin d'indiquer des contraintes sur les temps d'exécution.

Une propriété intéressante est que le model-checking de TCTL est décidable sur les automates temporisés [Alur 94, Henzinger 94]. La traduction d'un TPN borné vers un TA temporellement bisimilaire proposée dans [Cassez 06b] prouve également que le model-checking de TCTL est décidable pour les TPNs bornés. Les auteurs de [Cassez 06b, Boucheneb 09] définissent justement une logique TCTL adaptée au contexte des réseaux de Petri temporels (et ITPN) appelée TPN-TCTL. Nous donnons ci-dessous la syntaxe et la sémantique de cette logique.

Définition 3.7 (TPN-TCTL) *La syntaxe des formules TPN-TCTL est donnée inductivement par la grammaire suivante :*

$$\varphi := \mathcal{P} \mid \neg\varphi \mid \varphi \Rightarrow \psi \mid \exists\varphi \; \mathcal{U}_I \; \varphi \mid \forall\varphi \; \mathcal{U}_I \; \varphi$$

où $I \in \mathcal{I}(\mathbb{Q}^+)$ est un intervalle temporel, $\mathcal{P} \in PR$, et $PR = \{\mathcal{P} \mid \mathcal{P} : M \to \{\mathbf{true}, \mathbf{false}\}\}$ est l'ensemble des propositions sur les marquages du réseau.

On utilise les abréviations suivantes $\exists\Diamond_I\varphi = \exists\mathbf{true} \; \mathcal{U}_I \; \varphi$, $\forall\Diamond_I\varphi = \forall\mathbf{true} \; \mathcal{U}_I \; \varphi$, $\exists\Box_I\varphi = \neg\forall\Diamond_I\neg\varphi$ et $\forall\Box_I\varphi = \neg\exists\Diamond_I\neg\varphi$.

On définit également la formule de réponse bornée *par $\varphi \rightsquigarrow_I \psi = \forall\Box(\varphi \Rightarrow \forall\Diamond_I\psi)$.*

Les formules TPN-TCTL sont interprétées sur les états d'un modèle $\mathcal{M} = (\mathcal{S}_\mathcal{N}, \mathcal{V})$, où $\mathcal{S}_\mathcal{N}$ est l'espace d'états du TPN (ou de l'ITPN) analysé et $\mathcal{V} : \mathcal{S}_\mathcal{N} \to 2^{PR}$ est une fonction qui évalue le marquage d'un état $q = (M, I)$, telle que $\mathcal{V}(q) = \{\mathcal{P} \in PR \mid \mathcal{P}(M) = true\}$.

Soit $q \in \mathcal{S}_\mathcal{N}$ un état et $\rho \in \pi(q)$ une exécution maximale partant de q, telle que $\rho = q_1 \xrightarrow{(d_1,t_1)} q_2 \xrightarrow{(d_2,t_2)} \cdots$ (donc avec $q_1 = q$). On définit $\rho^* : \mathbb{R}^+ \to \mathcal{S}_\mathcal{N}$ par $\rho^*(r) = q_{i+1} + \delta$ si $r = \sum_{j=0}^{i} d_j + \delta$, avec $i \geq 0$, $d_0 = 0$ et $0 \leq \delta < d_{i+1}$. $\rho^*(r)$ est donc l'état atteint dans la séquence ρ après un délai de r unités de temps.

Définition 3.8 (Sémantique de TPN-TCTL) *Étant donné un TPN (ou un ITPN) $\mathcal{N}$ et le modèle $\mathcal{M} = (\mathcal{S}_\mathcal{N}, \mathcal{V})$, la valeur de vérité d'une formule TPN-TCTL pour un état $q \in \mathcal{S}_\mathcal{N}$ est donnée par :*

- $q \models \mathcal{P}$ *ssi* $\mathcal{P} \in \mathcal{V}(q)$,
- $q \models \neg\varphi$ *ssi* $q \not\models \varphi$,
- $q \models \varphi \Rightarrow \psi$ *ssi* $q \not\models \varphi \;\vee\; q \models \psi$,
- $q \models \exists\varphi \; \mathcal{U}_I \; \psi$ *ssi* $\exists\rho \in \pi(q), \; \exists r \in I$ *t.q.* $\rho^*(r) \models \psi$ *et* $\forall r' < r, \; \rho^*(r') \models \varphi$
- $q \models \forall\varphi \; \mathcal{U}_I \; \psi$ *ssi* $\forall\rho \in \pi(q), \; \exists r \in I, \;$ *t.q.* $\rho^*(r) \models \psi$ *et* $\forall r' < r, \; \rho^*(r') \models \varphi$

Étant donné un modèle $\mathcal{M} = (\mathcal{S}_\mathcal{N}, \mathcal{V})$, pour une proposition sur les marquages $\mathcal{P} \in PR$ et un état $q = (M, I) \in \mathcal{S}_\mathcal{N}$, on utilise la notation $M \models \mathcal{P}$ si $\mathcal{P} \in \mathcal{V}(q)$ et $M \not\models \mathcal{P}$ si $\mathcal{P} \notin \mathcal{V}(q)$.

Finalement, un TPN $\mathcal{N}$ satisfait une formule TPN-TCTL ϕ si et seulement si $q_0 \models \phi$, et nous notons alors $\mathcal{N} \models \phi$.

Exemple 3.4 *Sur le réseau de la figure 3.3, la propriété précédemment décrite par un observateur « t_3 est toujours tirée strictement moins de 4 unités de temps après t_1 » peut être spécifiée en TCTL par la formule suivante :*

$$(M(p_3) > 0) \rightsquigarrow_{[0,4[} (M(p_5) > 0)$$

3.4 Méthode du graphe des classes d'états

L'espace d'états d'un réseau de Petri temporel (ou d'un réseau de Petri à chronomètres) qui est décrit par le système de transitions définissant la sémantique du réseau est en général infini puisqu'il existe *a priori* une infinité d'états temporels accessibles. Pour représenter et analyser l'ensemble des états temporels accessibles il est donc nécessaire d'avoir recours à une méthode d'abstraction de l'information temporelle afin de pouvoir regrouper les états dans un ensemble fini de classes d'équivalence.

Des méthodes symboliques de représentation de l'espace d'états sont donc utilisées. Le *graphe des régions* [Alur 94] est une méthode théorique pour calculer l'espace d'états d'un automate temporisé dans laquelle les valeurs des horloges sont regroupées dans un ensemble de classes d'équivalence : les *régions*. En pratique, le nombre de régions étant trop important pour effectuer une analyse efficace, les régions sont regroupées dans un ensemble de *zones*. Une zone est une union convexe de regions et peut être représentée par une DBM. Des algorithmes efficaces utilisant cette méthode pour vérifier des propriétés CTL et TCTL sur les automates temporisés ont été implémentés dans les outils Uppaal [Larsen 97] et Kronos [Yovine 97].

Dans les réseaux de Petri temporels, une méthode classique de représentation de l'espace d'états est le *graphe des classes d'états* [Berthomieu 83, Berthomieu 91]. Une classe (M, D) associe un marquage M et un domaine temporel D représenté par un ensemble d'inégalités sur des variables. Contrairement aux régions et aux zones dans les TA, les variables représentées dans le graphe des classes d'états ne sont pas les horloges des transitions mais les intervalles temporels de tir des transitions sensibilisées. Le graphe des classes d'états préserve les marquages et le langage non temporel de l'espace d'états. Il permet donc de vérifier des propriétés d'accessibilités de marquage et des propriétés LTL. Par contre, il ne permet pas de vérifier des propriétés de branchement de type CTL.

Afin de remédier à cette limitation, un raffinement de la méthode a été proposé dans [Yoneda 98] sous la forme d'un graphe des *classes d'états atomiques*. Les auteurs utilisent pour cela un découpage plus poussé des classes d'états en rajoutant des contraintes linéaires, de sorte que chaque état d'une classe atomique possède un successeur dans toutes les classes suivantes. Grâce à cette méthode ils sont capables de vérifier des propriétés CTL* sur les TPN, mais avec comme limitation que les intervalles temporels des transitions soient bornés. Une méthode alternative de construction des classes atomiques a été alors proposée dans

[Berthomieu 03] ; elle permet de vérifier des propriétés CTL* plus efficacement et sans limitation sur les intervalles temporels.

Une méthode alternative au graphe des classes d'états pour représenter l'espace d'états d'un TPN est proposée dans [Gardey 03, Gardey 05a] sous la forme d'un *graphe des zones*. La méthode est donc une adaptation des méthodes utilisées dans les TA. Ainsi, contrairement au graphe des classes, ce sont directement les horloges associées aux transitions qui sont encodées dans les zones. Cela permet de vérifier directement des propriétés temporelles quantitatives mais pas des propriétés CTL*. Comme pour les TA, un inconvénient de la méthode est le recours nécessaire à des méthodes d'approximation (k-approximation ou k_x-approximation [Gardey 05a]) dans le cas où l'infini est utilisée dans les bornes des intervalles temporels.

Dans le cas des réseaux de Petri à chronomètres, la méthode du graphe des classes d'états a été étendue aux ITPN dans [Roux 04]. C'est cette méthode que nous utiliserons par la suite et que l'on présente donc ici directement pour les réseaux de Petri avec arcs inhibiteurs (la méthode s'applique de manière similaire sur les TPN, il n'y a simplement pas de transitions inhibées).

3.4.1 Classes d'états

Définition 3.9 (Classe d'états) *Une* classe d'états C *d'un ITPN est une paire* (M, D)*, où* M *est un marquage du réseau et* D *un* domaine de tir *représenté par un polyèdre convexe de* $\mathbb{R}^{+T}$.

Un point $x = (\theta_1, \theta_2, \ldots, \theta_n) \in D$ *du domaine de tir est constitué des valeurs des variables* $\theta_1, \theta_2, \ldots, \theta_n$*, qui donnent les dates de tir dans* C *des transitions sensibilisées par* M [2].

Une classe d'états regroupe l'ensemble des états accessibles par une même séquence de transitions non temporisée s. En particulier, ces états possèdent le même marquage. Elle ne décrit cependant que les états à l'entrée dans la classe, c'est-à-dire les états qui sont accessibles par une exécution $\rho = q_0 \xrightarrow{(d_0,t_0)} q_1 \xrightarrow{(d_1,t_1)} \cdots \xrightarrow{(d_n,t_n)} q$ tels que $s = t_0; t_1; \cdots; t_n$. Le domaine de tir correspond alors à l'union des intervalles temporels des états agrégés dans la classe. Ainsi, pour une classe d'états $C = (M, D)$, si $q \in C$ alors $q = (M, I)$ avec $I \subseteq D$. La réciproque est fausse car deux classes d'états accessibles par deux séquences de transitions différentes peuvent posséder le même marquage et le même domaine de tir (ces classes seront cependant regroupées grâce à la relation d'égalité définie en 3.12).

Dans le cas des réseaux de Petri T-temporels les inéquations définissant les domaines de tir sont de formes plus simples, ce qui permet de les encoder en utilisant des DBM. La manipulation des domaines de tir est alors beaucoup plus efficace. Hélas, cette propriété n'est pas valable dans le cas des réseaux à chronomètres où des inéquations ne correspondant pas à la forme des DBM sont introduites ; il est donc dans ce cas indispensable de conserver des polyèdres généraux.

[2] En pratique seules les variables associées aux transitions sensibilisées sont conservées dans D. Cela facilite le calcul sur les polyèdres dont la complexité croît avec le nombre de variables.

3.4.2 Calcul du graphe des classes d'états

Avant de présenter la méthode de calcul du graphe des classes d'états, il est nécessaire d'étendre la définition de transition tirable depuis un état, à celle de transition tirable depuis une classe.

Définition 3.10 (Transition tirable depuis une classe d'états) *Soit une classe d'états $C = (M, D)$ d'un ITPN, une transition t_i est dite* tirable *depuis C, si elle est sensibilisée et non inhibée par M, et s'il existe une solution $(\theta_1, \ldots, \theta_n) \in D$, telle que $\forall j \in [1..n]$ t.q. $j \neq i$ et $t_j \in \mathsf{enabled}(M)$ et $t_j \notin \mathsf{inhibited}(M)$, $\theta_i \leq \theta_j$.*
Nous notons cela : $t_i \in \mathsf{firable}(C)$.

Une transition t_i est tirable depuis une classe d'états C s'il existe au moins un état dans cette classe pour lequel t_i est tirable.

Définition 3.11 (Successeur d'une classe d'états) *Soit une classe d'états $C = (M, D)$ et une transition t_f tirable dans C. La classe d'états $C' = (M', D')$, successeur de C par le tir de t_f, ce que l'on note $C' = \mathsf{succ}(C, t_f)$, est calculée de la manière suivante :*

- *$M' = M - {}^\bullet t_f + t_f^\bullet$*
- *D' est calculé à partir de D par les étapes suivantes, et noté $\mathsf{next}(D, t_f)$:*
  1. *intersection avec les contraintes de franchissabilité : $\forall j$ t.q. t_j est* active, *$\theta_f \leq \theta_j$*
  2. *substitution des variables pour toutes les transitions actives t_j : $\theta_j = \theta_f + \theta'_j$,*
  3. *élimination (en utilisant par exemple la méthode de Fourier-Motzkin) de toutes les variables relatives aux transitions désensibilisées par le tir de t_f (pour les développements théoriques présentés dans ce manuscrit ces variables sont conservées mais fixées à zéro),*
  4. *ajout des inéquations relatives aux transitions nouvellement sensibilisées :*

$$\forall t_k \in \uparrow \mathsf{enabled}(M, t_f),\ {}^{\uparrow}I_s(t_k) \leq \theta'_k \leq I_s(t_k)^{\downarrow}$$

La transition t_f étant supposée tirable depuis C, le polyèdre contraint par les inégalités $\theta_f \leq \theta_j$ est nécessairement non vide. Cependant ces contraintes entraînent que seuls les états de C pour lesquels t_f est tirable possèdent un successeur dans C'.

La substitution des variables correspond à un changement d'origine temporelle pour les transitions actives : la nouvelle origine du temps pour les dates de tir dans C' est la date de tir de t_f. Les contraintes de franchissabilité peuvent d'ailleurs être ajoutées de manière équivalente après le changement de variables avec les inéquations suivantes : $\forall j \neq f,\ \theta'_j \geq 0$.

Calcul pour un point Soit une classe d'états $C = (M, D)$ d'un ITPN, pour un point $x = (\theta_1, \ldots, \theta_n)$ du domaine de tir D on calcule formellement ses successeurs après le tir d'une transition t_f tirable depuis $(M, \{x\})$:

$$\mathsf{next}(\{x\}, t_f) = \left\{ (\theta'_1, \ldots, \theta'_n),\ \forall i \in [1..n] \;\middle|\; \begin{array}{l} \theta'_i \in I_s(t_i) \text{ si } \uparrow \mathsf{enabled}(t_i, M, t_f) \\ \theta'_i = \theta_i \text{ si } t_i \in \mathsf{enabled}(M) \\ \quad \text{et } t_i \in \mathsf{inhibited}(M) \\ \quad \text{et non } \uparrow \mathsf{enabled}(t_i, M, t_f) \\ \theta'_i = 0 \text{ si } t_i \notin \mathsf{enabled}(M) \\ \theta'_i = \theta_i - \theta_f \text{ sinon} \end{array} \right\}$$

Cet opérateur next est étendu à une union finie ou infinie de points pour définir formellement $\mathsf{next}(D, t_f)$.

Graphe des classes d'états Le graphe des classes d'états est alors généré itérativement en appliquant la fonction définissant les successeurs d'une classe d'états.

Étant donné un ITPN $\mathcal{N}$, on calcule le graphe des classes d'états de $\mathcal{N}$ en construisant le système de transitions $\mathcal{G}(\mathcal{N}) = \langle \mathcal{C}, C_0, \twoheadrightarrow, \rangle$ tel que :

- $C_0 = (M_0, D_0)$ est la classe initiale : M_0 est le marquage initial de $\mathcal{N}$, D_0 est défini par $D_0 = \{\theta_k \in I_s(t_k) \mid t_k \in \mathsf{enabled}\,(M_0)\}$,
- $C \overset{t}{\twoheadrightarrow} C'$ ssi $\begin{cases} t \in \mathsf{firable}\,(C)\,, \\ C' = \mathsf{succ}(C, t), \end{cases}$
- $\mathcal{C} = \{C \mid C_0 \twoheadrightarrow^* C\}$, où $\twoheadrightarrow^*$ est la fermeture réflexive et transitive de $\twoheadrightarrow$.

Défini ainsi, il n'y *a priori* aucune raison pour que le graphe des classes d'états d'un ITPN soit fini, à moins que toutes les exécutions se terminent dans une classe sans successeur. On utilise donc un critère de convergence en définissant une relation d'égalité ou d'inclusion entre classes afin de pouvoir reboucler sur les classes déjà construites. Le choix de la relation de convergence dépend du type de propriété que l'on souhaite préserver : l'accessibilité avec l'inclusion, les propriétés LTL avec l'égalité.

Définition 3.12 (Égalité 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 $C \equiv C'$.*

Définition 3.13 (Inclusion de classes d'états) *Une classe d'états paramétrée $C = (M, D)$ est dite incluse dans une classe $C' = (M', D')$ si $M = M'$ et $D \subseteq D'$. Nous notons $C \subseteq C'$.*

Le *graphe des classes d'états* $\mathcal{G}'(\mathcal{N})$ d'un ITPN est alors défini comme le quotient de $\mathcal{G}$ par la relation d'équivalence $\mathcal{R}$, qui est soit l'égalité, soit l'inclusion de classes d'états (sauf précision supplémentaire, lorsque nous parlerons du graphe des classes d'états nous considérerons celui construit avec la relation égalité) :

$$\mathcal{G}'(\mathcal{N}) = \mathcal{G}(\mathcal{N})/\mathcal{R}$$

Cependant même avec ce critère, puisque le problème de l'accessibilité de marquage dans un réseau de Petri à chronomètres n'est pas décidable [Lime 04a, Berthomieu 07], le graphe des classes d'états d'un ITPN n'est *a priori* pas fini.

Dans le cas des réseaux de Petri T-temporels par contre, le théorème suivant est valable :

Théorème 3.1 ([Berthomieu 83, Berthomieu 91]) *Le nombre de classes du graphe des classes d'états d'un réseau de Petri T-temporel $\mathcal{N}$ est fini si et seulement si $\mathcal{N}$ est borné.*

Comme évoqué en introduction, le graphe des classes d'états permet d'obtenir les marquages accessibles du réseau de Petri, et permet de vérifier des propriétés de logique temporelle LTL si l'on converge par inclusion. Il ne permet cependant pas de vérifier des propriétés de branchement du type CTL. En effet, comme précisé précédemment, une classe C' est le successeur d'une classe C s'il existe au moins un état dans C qui a un successeur dans C', mais ce n'est donc pas le cas pour tous les états de C. Les méthodes proposées dans

[Yoneda 98, Berthomieu 03] modifient le graphe des classes d'états de sorte que justement tous les états de C possèdent un successeur dans C'. Cela permet de vérifier des propriétés du type CTL*, mais ce n'est pas encore suffisant pour vérifier des propriété du type TCTL.

Toutefois, des méthodes ont été proposées pour vérifier un sous-ensemble de formules TCTL avec le graphe des classes [Hadjidj 06] et également avec le graphe des zones [Gardey 05a, Boucheneb 09]. Elles seront évoquées dans la suite.

Exemple 3.5 *On construit sur la figure 3.5 le graphe des classes du réseau de Petri temporel avec arcs inhibiteurs de la figure 3.3. Les 5 classes sont dessinées par des noeuds et sont reliées entre elles par des transitions. On écrit à coté de chaque classe le marquage et le domaine de tir de la classe. Un schéma classique apparaît dans ce graphe : l'entrelacement entre t_3 et t_4 figuré par un losange, lorsque l'on choisit de tirer t_4 d'abord puis t_3 ou l'opposé. On s'aperçoit en construisant le graphe des classes d'états de ce réseau que la place p_4 n'est jamais marquée, ce qui prouve qu'aucun marquage M tel que $M(p_4) > 0$ n'est accessible.*

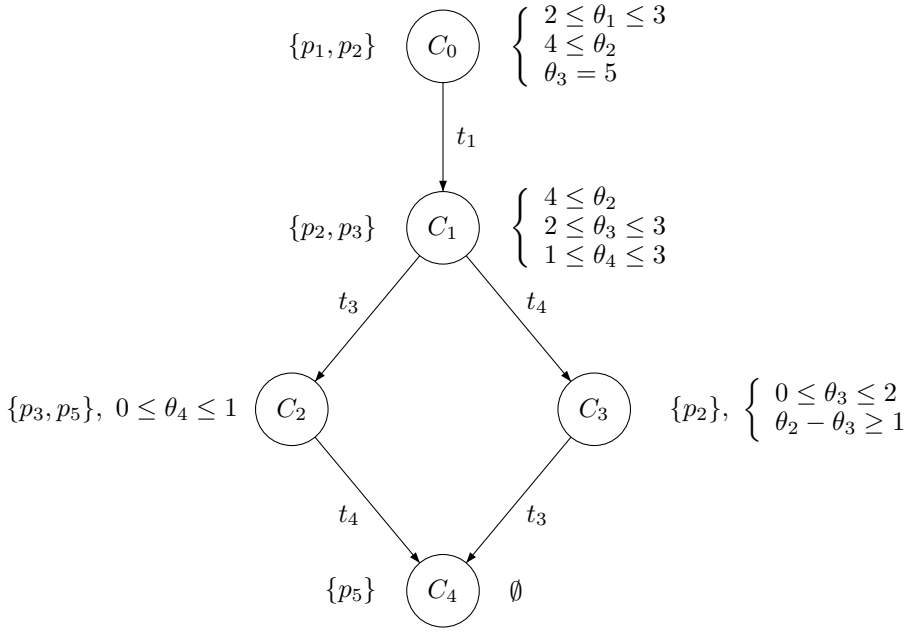


FIG. 3.5: Graphe des classes d'états du réseau ITPN de la figure 3.3

Deuxième partie

Model-checking paramétré

Chapitre 4

Model-checking paramétré sur les réseaux de Petri temporels

Résumé : *Ce chapitre présente l'une des contributions de la thèse. Nous introduisons le problème du model-checking paramétré à la frontière du contrôle et de la vérification : le but est de synthétiser des valeurs possibles pour les paramètres du modèle. Pour cela les techniques de model-checking classiques sont étendues aux réseaux de Petri à chronomètres paramétrés. Nous présentons ainsi des semi-algorithmes de vérification de formules TCTL paramétrées qui sont basés sur l'exploration du graphe des classes d'états paramétrées du réseau.*

Ces travaux ont fait l'objet d'une publication à la conférence internationale FORMATS (Formal Modelling and Analysis of Timed Systems) en 2008 [Traonouez 08] ainsi qu'une publication dans la revue JUCS (Journal of Universal Computer Science) [Traonouez 09].

Sommaire

4.1 Problématique du model-checking paramétré

Le model-checking est une technique de vérification formelle maintenant très répandue. Le principe général est d'explorer tout ou partie de l'espace d'états du modèle pour rechercher les états vérifiant une spécification donnée. La technique a connu d'importantes avancées et est devenue de plus en plus efficace, notamment grâce à l'utilisation de structure de données évoluées telles que les BDD et leurs dérivés. Elle a ensuite été étendue aux modèles temporisés. Dans ces modèles cependant, si une sémantique de temps dense est considérée, l'espace d'états du système est généralement infini. Des techniques de model-checking symboliques ont donc été introduites afin de regrouper les états en ensemble fini de classes d'états ou de régions.

Lors de l'utilisation du model-checking dans les modèles temporisés, il est nécessaire de connaître entièrement le système et son environnement. Il est en particulier nécessaire de connaître toutes les caractéristiques temporelles du système, ce qui n'est possible qu'après la phase de conception du système. Par ailleurs, la technique doit en général être appliquée sur l'ensemble du système, et non sur seulement une partie du système. Une première conséquence est d'augmenter la complexité de la vérification, ce qui peut amener à une explosion combinatoire rendant l'analyse impossible. Et si malgré cela, l'analyse prouve que des spécifications temporelles ne sont pas vérifiées, il sera nécessaire de reprendre entièrement ce processus complexe de conception et de vérification.

Or il se trouve que pendant la phase de conception d'un système, en utilisant les méthodes d'ingénierie dirigées par les modèles, il est fréquent d'introduire des paramètres afin de laisser libres certaines spécifications. On peut notamment paramétrer les caractéristiques temporelles du système pendant les premières phases de la conception. Les valeurs réelles de ces paramètres ne seront instanciées qu'à la fin de la conception lors de la projection du modèle sur une architecture matérielle.

De la même manière, on est amené à décomposer le système en sous-parties modélisées séparément par des sous-modèles ou modules. Des paramètres peuvent alors être utilisés pour spécifier des entrées et des sorties de ces modules. Pour vérifier le système les modules doivent être connectés et les paramètres éliminés.

Un outil comme Uppaal [Larsen 97] propose déjà comme fonctionnalités l'utilisation de paramètres et la définition de modules. Cependant, pour vérifier une propriété sur le système il sera nécessaire d'instancier ces paramètres, et d'utiliser comme modèle la composition parallèle de l'ensemble des modules.

Si l'on souhaite déterminer des valeurs pour les paramètres tout en vérifiant des spécifications données on est obligé de procéder par dichotomie avec une approche pas à pas. On essaye par exemple d'instancier les paramètres à une certaine valeur, on vérifie par model-checking les propritétés recherchées, puis on fait varier les paramètres jusqu'à obtenir un ensemble de valeurs satisfaisantes. Cette méthode est fastidieuse et demande d'effectuer à de nombreuses reprises le processus de vérification. Elle est presque inutilisable si plusieurs paramètres indépendants doivent être déterminés.

Contrôle vs. vérification

Le *problème de la vérification* consiste à déterminer si un système S vérifie une spécification ϕ. Il peut être résolu en construisant un modèle $\mathcal{S}$ de S et en vérifiant par model-checking

que $\mathcal{S}$ vérifie la spécification ϕ décrite par un propriété φ que l'on exprime dans une logique appropriée. On note ceci en général $\mathcal{S} \models \varphi$ ce qui constitue le problème du model-checking.

Le *problème du contrôle* suppose au contraire que le système S est ouvert, c'est-à-dire que l'on peut restreindre son comportement. Ainsi, certains évènements de S vont être contrôlables et d'autres non. Le problème est alors de déterminer à partir d'un modèle $\mathcal{S}$ de S et d'une spécification ϕ décrite par une propriété φ s'il existe un contrôleur C dont le modèle serait $\mathcal{C}$ et tel que $\mathcal{S} \times \mathcal{C} \models \varphi$. Le problème associé de la synthèse d'un contrôleur cherche à calculer le contrôleur C.

Model-checking paramétré

Le problème du model-checking paramétré qui conduit à la synthèse de paramètres se situe à la frontière entre la vérification et le contrôle. Étant donné un modèle paramétré $\mathcal{S}$ et une propriété paramétrée φ, on cherche à déterminer s'il existe une valuation ν des paramètres telle que $\mathcal{S}$ satisfait φ pour cette valuation. On note $[\![\mathcal{S}]\!]_\nu$ le modèle non paramétré obtenu en remplaçant dans $\mathcal{S}$ les paramètres par leur valeur dans ν et $[\![\varphi]\!]_\nu$ la propriété non paramétrée obtenue pour la même valuation. Le problème du model-checking paramétré peut alors se noter : $\exists \nu, [\![\mathcal{S}]\!]_\nu \models [\![\varphi]\!]_\nu$.

Le problème associé de la synthèse des paramètres cherche quant à lui à déterminer l'ensemble des valuations Γ des paramètres qui permettent de vérifier le problème, c'est-à-dire $\forall \nu \in \Gamma$, $[\![S]\!]_\nu \models [\![\phi]\!]_\nu$.

Le model-checking paramétré suppose donc un système partiellement contrôlable par l'intermédiaire de variables pouvant être ajustées : les paramètres. La méthode utilisée se rapprochera par contre du model-checking classique et des techniques déjà existantes dans ce domaine, mais elles seront étendues pour gérer les paramètres et déterminer *in fine* des valuations satisfaisantes.

La technique pourra alors être utilisée au cours de la conception du système pour vérifier des spécifications sur un modèle paramétré du système ou d'une sous-partie du système et déterminer des valeurs possibles pour les paramètres.

4.2 État de l'art

4.2.1 Model-checking de réseaux de Petri temporels

La vérification des réseaux de Petri temporels nécessite d'utiliser une abstraction de l'espace d'états telle que le graphe des classes ou le graphe des zones présentés au chapitre 3. Ces méthodes permettent de vérifier dans le modèle l'accessibilité d'états ou des formules LTL, et des propriétés plus évoluées en utilisant des observateurs.

Une autre approche pour vérifier des propriétés sur des réseaux de Petri temporels consiste à les traduire en automates temporisés ayant le même comportement. Les méthodes et les outils existant pour les TA peuvent alors être utilisés pour vérifier les TPN. Le model-checking de formules TCTL est notamment décidable [Alur 90]. Pour cela plusieurs méthodes sont utilisables. Une première catégorie de méthodes sont des traductions structurelles, telle que celle présentée dans [Cassez 06b] qui propose une traduction structurelle d'un TPN vers un TA temporellement bisimilaire. Les autres méthodes sont basées sur le calcul de l'espace d'états.

On peut ainsi calculer un automate des classes d'états [Lime 06] temporellement bisimilaire. La méthode du graphe des zones permet également d'obtenir un automate temporisé des marquages équivalent au TPN initial [Gardey 06].

Mais il est également possible de vérifier des propriétés TCTL directement à partir du TPN sans passer par une traduction. Ainsi, les auteurs de [Gardey 05a, Boucheneb 09] proposent une méthode pour vérifier des formules TCTL en utilisant le graphe des zones. Ils appliquent un principe utilisé sur les TA qui consiste à vérifier des formules TCTL en les transformant en formules CTL [Tripakis 01, Penczek 04]. Il faut pour cela ajouter une transition spéciale au réseau pour mesurer le temps écoulé. Mais puisque le graphe des zones ne préserve pas les propriétés de branchement CTL, il faut également le raffiner pour considérer un graphe des zones atomiques. La méthode nécessite donc de construire entièrement ce graphe pour chaque formule vérifiée.

En conséquence, les auteurs présentent également une méthode pour vérifier cette fois directement « à la volée » sur le graphe des zones un sous-ensemble de formules TCTL. Ainsi, la construction de l'espace d'états est réalisée au fur et à mesure de l'exploration et est arrêtée dès que le résultat est trouvé.

Ce sous-ensemble de formules TCTL considérées est appelé TPN-TCTL$_{\mathcal{S}}$ et est défini avec les formules suivantes :

$$\text{TPN-TCTL}_{\mathcal{S}} := \exists\varphi\ \mathcal{U}_I\ \psi \mid \forall\varphi\ \mathcal{U}_I\ \psi \mid \exists\Diamond_I\varphi \mid \forall\Diamond_I\varphi \mid \exists\Box_I\varphi \mid \forall\Box_I\varphi \mid \varphi \rightsquigarrow_{I_r} \psi$$

où φ, ψ sont des propositions sur les marquages du réseau de Petri, et I, I_r des intervalles temporels tels que $I = [a, b]$ ou $I = [a, \infty[$, et $I_r = [0, b]$ ou $I_r = [0, \infty[$, avec $a, b \in \mathbb{Q}^+$. Ainsi défini, ce sous-ensemble restreint l'utilisation de la récursivité dans les formules (elle est interdite sauf en utilisant une formule du type réponse bornée qui intègre par définition un niveau de récursivité). Ces formules seront plus simples à analyser puisqu'aucun retour sur trace ne sera nécessaire : des algorithmes de vérification « à la volée » peuvent donc être utilisés. De plus ce sous-ensemble reste suffisant pour vérifier la plupart des propriétés recherchées du type accessibilité, sécurité ou vivacité.

Enfin, dans [Hadjidj 06] les auteurs présentent également une méthode de vérification de ce sous-ensemble de formules TCTL sur les TPN, basée cette fois sur le graphe des classes. Pour vérifier les bornes temporelles des formules TCTL les auteurs utilisent une méthode de type observateur, en ajoutant deux transitions au réseau pour observer le début et la fin de l'intervalle temporel de la formule.

4.2.2 Travaux de model-checking paramétré

Le model-checking paramétré de systèmes temps réel a déjà été étudié par les auteurs de [Alur 93]. Ils introduisent pour cela des automates temporisés paramétrés (PTA) dans lesquel des paramètres peuvent être utilisés dans les gardes. Ils s'intéressent à l'accessibilité d'états dans ces modèles paramétrés, et prouvent que dans le cas général, le problème de l'existence d'une valuation des paramètres permettant d'atteindre un état (ou problème du vide) n'est pas décidable. Ce problème devient décidable si l'on considère des automates ne contenant qu'une seule horloge contrainte par des paramètres.

Sur les automates temporisés paramétrés, les auteurs de [Annichini 00] présentent une technique d'analyse de l'accessibilité qui utilise des DBM paramétrées pour représenter l'es-

pace d'états, ainsi qu'une technique d'extrapolation qui permet de deviner le résultat de l'itération des boucles un nombre arbitraire de fois.

Dans [Hune 01] les auteurs étudient les PTA en utilisant également des DBM paramétrées. Ils trouvent une sous-classe de ces automates qu'ils nomment *L/U automata* pour laquelle le problème du vide est décidable, et ils présentent un algorithme de model-checking paramétré.

Une autre approche est présentée dans [André 08]. À partir d'un PTA et d'une valuation des paramètres ν, le problème étudié est de déterminer des contraintes telles que pour toutes les valuations vérifiant ces contraintes le comportement non temporisé du PTA soit équivalent à celui obtenu pour la valuation ν.

Enfin, en temps discret, pour des PTA avec une seule horloge paramétrée, le model-checking de formules TCTL, qui peuvent également être paramétrées, est décidable avec la restriction de ne pas utiliser l'égalité dans les formules [Bruyère 07].

Le modèle des automates hybrides permet également d'effectuer une analyse paramétrée. Les paramètres sont dans ce cas modélisés par des horloges constantes. Les algorithmes d'exploration de l'espace d'états des automates hybrides ont été ainsi adaptés pour effectuer une analyse paramétrée, ce qui est implémenté notamment dans l'outil HYTECH [Henzinger 97].

Une autre approche développée dans [Wang 96] se focalise sur la vérification de formules TCTL paramétrées sur des automates à horloges. Ils considèrent des paramètres non bornés qui peuvent prendre des valeurs entières. Dans ce cadre, le problème du model-checking paramétré est prouvé décidable.

Enfin, dans [Virbitskaite 99] l'approche précédente est utilisée dans les réseaux de Petri T-temporels paramétrés. Les auteurs considèrent cependant des paramètres bornés à valeurs entières. Ils peuvent alors construire un graphe des régions pour chaque valuation des paramètres, et appliquer la méthode de [Wang 96] sur ce graphe pour vérifier des formules TCTL paramétrées.

4.3 Réseaux de Petri temporels paramétrés avec arcs inhibiteurs

4.3.1 Syntaxe

On introduit des paramètres temporels dans le modèle des réseaux de Petri avec arcs inhibiteurs présenté au chapitre 3. On dispose pour cela d'un ensemble de paramètres temporels $Par = \{\lambda_1, \lambda_2, \ldots, \lambda_l\}$ que l'on peut utiliser pour remplacer les bornes temporelles des transitions. Dans la sémantique du modèle ainsi défini ces paramètres sont considérés comme étant des variables constantes dont les valeurs sont rationnelles.

Des contraintes initiales peuvent être ajoutées sur les paramètres. Ces contraintes linéaires forment le domaine $D_p \subseteq \mathbb{Q}^{+Par}$ des paramètres, représenté par un polyèdre convexe. Au minimum ces contraintes doivent spécifier que pour toutes les valeurs des paramètres dans D_p les bornes minimales des intervalles de tir des transitions sont inférieures à leur borne maximale. Mais des contraintes supplémentaires peuvent évidemment être spécifiées afin d'enrichir la modélisation.

Un *intervalle temporel paramétré* est une fonction $J : D_p \to \mathcal{I}(\mathbb{Q}^+)$, qui associe à une valuation des paramètres $\nu \in D_p$ un $\mathbb{Q}^+$-intervalle. J est composée de deux fonctions ${}^{\uparrow}J$ et

$J^{\downarrow}$ qui sont respectivement la borne minimale et la borne maximale de l'intervalle. On se restreint à des intervalles paramétrés où ${}^{\uparrow}J$ est une expression linéaire sur les paramètres, et $J^{\downarrow}$ est soit une expression linéaire, soit égal à l'infini. L'ensemble de ces intervalles paramétrés sur Par est noté $\mathcal{J}(Par)$.

Lorsque l'intervalle temporel paramétré est associé à une transition t d'un réseau de Petri temporel paramétré, on pourra à nouveau noter $\mathsf{eft}(t)$ et $\mathsf{lft}(t)$ les bornes minimales et maximales de l'intervalle. Elles correspondent alors à des expressions linéaires sur les paramètres.

Définition 4.1 (Réseau de Petri temporel paramétré avec arcs inhibiteurs) *Un* réseau de Petri temporel paramétré avec arcs inhibiteurs (PITPN) *est un 9-uplet* $\mathcal{N} = \langle P, T, Par, {}^{\bullet}(.), (.)^{\bullet}, {}^{\circ}(.), M_0, J_s, D_p \rangle$*, tel que :*

- $\langle P, T, {}^{\bullet}(.), (.)^{\bullet}, M_0 \rangle$ *est un réseau de Petri,*
- $Par = \{\lambda_1, \lambda_2, \ldots, \lambda_l\}$ *est un ensemble fini non vide de* paramètres *tel que* P*,* T *et* Par *sont disjoints deux à deux,*
- ${}^{\circ}(.) \in (\mathbb{N}^P)^T$ *est la* fonction d'inhibition*,*
- $J_s \in (\mathcal{J}(Par))^T$ *est la fonction associant un* intervalle temporel paramétré *à chaque transition,*
- D_p *est un polyèdre convexe non vide de* $\mathbb{Q}^{+Par}$ *décrivant le* domaine des paramètres*.*

Exemple 4.1 *Un exemple de PITPN est donné sur la figure 4.1. Il inclut 3 paramètres* a*,* b *et* c*. On rajoute les contraintes suivantes sur les paramètres :* $b < c \leq 5$*, le domaine des paramètres est donc :*

$$D_p = \left\{ \begin{array}{l} 0 \leq a \leq b \\ 1 \leq b < c \leq 5 \end{array} \right\}$$

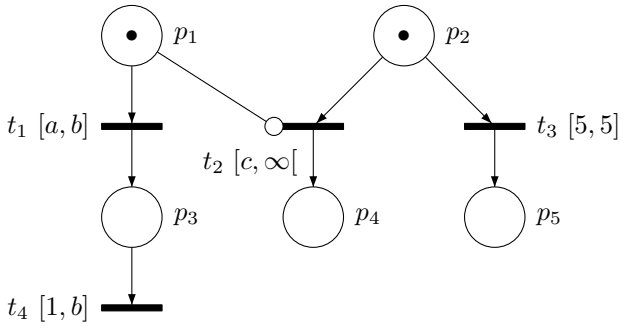


FIG. 4.1: Réseau de Petri temporel paramétré avec arcs inhibiteurs

4.3.2 Sémantique

La sémantique d'un réseau de Petri temporel paramétré avec arcs inhibiteurs $\mathcal{N}$ est définie pour une valuation $\nu \in D_p$ des paramètres comme le réseau ITPN non paramétré obtenu en remplaçant dans $\mathcal{N}$ tous les paramètres par leur valeur.

Définition 4.2 (Sémantique d'un PITPN) *Étant donné un réseau de Petri temporel paramétré avec arcs inhibiteurs* $\mathcal{N} = \langle P, T, Par, {}^{\bullet}(.), (.)^{\bullet}, {}^{\circ}(.), M_0, J_s, D_p \rangle$*, et une valuation* $\nu \in$

D_p, la sémantique $[\![\mathcal{N}]\!]_\nu = \langle P, T, {}^\bullet(.), (.)^\bullet, {}^\circ(.), M_0, I_s \rangle$ de $\mathcal{N}$ est un ITPN tel que :

$$\forall t \in T,\ I_s(t) = J_s(t)(\nu)$$

On dira qu'un PITPN est zénon s'il existe une valuation $\nu \in D_p$ telle que $[\![\mathcal{N}]\!]_\nu$ est zénon. Dans la suite on se restreint à des PITPN non zénons.

Exemple 4.2 *Si l'on considère le PITPN de la figure 4.1 et la valuation $\nu = (2, 3, 4) \in D_p$ des paramètres (a, b, c), alors la sémantique $[\![\mathcal{N}]\!]_\nu$ correspond au réseau ITPN non paramétré présenté sur la figure 3.3, dont une exécution est présentée dans l'exemple 3.2*

4.4 Indécidabilité

Nous donnons dans cette partie des résultats concernant la décidabilité des problèmes dans les réseaux de Petri temporels paramétrés. Le principal problème est celui du *vide* ou de l'*accessibilité paramétrée*, c'est-à-dire l'existence ou non d'une valuation des paramètres permettant l'accessibilité d'un marquage ou d'un état. Ce problème a déjà été étudié dans le cas des automates temporisés paramétrés et prouvé indécidable dans le cas général.

Pour un réseau de Petri temporel paramétré $\mathcal{N}$ (avec ou sans arcs inhibiteurs), $[\![\mathcal{N}]\!]_\nu$ est l'ITPN obtenu pour une valuation $\nu \in D_p$ et $\mathcal{S}_{[\![\mathcal{N}]\!]_\nu} = \langle Q, q_0, \rightarrow \rangle$ est la sémantique de $[\![\mathcal{N}]\!]_\nu$. On définit les problèmes paramétriques suivant :

- l'*accessibilité paramétrée de marquage* : étant donné un marquage M, « $\exists \nu \in D_p,\ \exists (M', I') \in Q,\ M = M'$ » ;
- la *bornitude* : « $\exists b \in \mathbb{N},\ \forall \nu \in D_p,\ \forall (M, I) \in Q,\ \forall p \in P,\ M(p) \leq b$ ».

On s'intéresse au problème de l'accessibilité paramétrée dans les PTPN bornés, le problème étant déjà connu indécidable dans le cas des ITPN bornés non paramétrés et des TPN non bornés (voir le chapitre 3).

Théorème 4.1 *L'accessibilité paramétrée de marquage est indécidable pour les réseaux de Petri temporels paramétrés.*

Preuve 4.1 (Théorème 4.1) *Une traduction structurelle et syntaxique d'un TA vers un TPN borné, qui préserve le langage temporisé accepté, est proposée dans [Bérard 05]. Elle peut être directement étendue pour traduire un PTA vers un PTPN acceptant le même langage. Or dans [Alur 93] les auteurs montrent l'indécidabilité du test du vide dans les PTA (c'est-à-dire l'absence de valuation des paramètres pour laquelle le langage accepté par le PTA est non vide), en réduisant le problème de l'arrêt d'une machine à 2 compteurs au problème du test du vide. Le PTA utilisé pour montrer ce résultat possède 3 horloges contraintes uniquement par des égalités. Il peut être traduit en PTPN acceptant le même langage par la méthode de [Bérard 05] (le PTA ne possédant que des contraintes d'égalités, il peut être traduit en PTPN borné avec intervalles fermés, tels que définis dans ce manuscrit). On prouve ainsi l'indécidabilité du problème du vide dans les PTPN bornés et donc de l'accessibilité paramétrée.*

4.5 Graphe des classes d'états paramétrées

Nous l'avons vu au chapitre 3, l'espace d'états d'un ITPN en temps dense même borné est en général infini. Il est donc nécessaire de recourir à des méthodes d'abstractions du temps en regroupant les états dans des classes d'équivalences. Une des approches classiques de représentation symbolique de l'espace d'états d'un TPN est le graphe des classes d'états [Berthomieu 91], qui peut également être appliqué sur les ITPN [Roux 04].

Cependant, dans le cas des réseaux paramétrés il existe également en général une infinité de valeurs possibles pour les paramètres. Nous devons donc de la même manière utiliser des représentations symboliques des domaines des paramètres. Dans les TPN ou les TA les domaines temporels des états abstraits peuvent être encodés efficacement à l'aide de DBM. Pour cette raison, dans les automates temporisés paramétrés proposés dans [Hune 01], les auteurs définissent des DBM paramétrées dans lesquelles ils encodent à la fois les domaines temporels des horloges et les domaines des paramètres. Lorsque l'on considère des réseaux de Petri à chronomètres la forme des domaines de tir dans le graphe des classes ne permet plus l'utilisation de DBM, mais uniquement de polyèdres généraux, moins efficaces à manipuler.

Pour représenter l'espace d'états d'un réseau de Petri temporel paramétré avec arcs inhibiteurs, on étend la méthode du graphe des classes d'états et on intègre les paramètres aux domaines de tir des classes. Par conséquent, dans les classes d'états paramétrées des PITPN on utilisera des polyèdres pour encoder les domaines de tir dans lesquels on trouvera alors deux types de variables : les dates de tir des transitions et les paramètres.

4.5.1 Classes d'états paramétrées

Reprenant la définition 3.9 des classes d'états, on définit les classes d'états paramétrées dans lesquelles sont ajoutés les paramètres.

Définition 4.3 (Classe d'états paramétrée) *Une* classe d'états paramétrée *C d'un PITPN est une paire (M, D), M est un marquage du réseau et D un* domaine de tir *représenté par un polyèdre convexe de $\mathbb{R}^{+(l+n)}$, où est l le nombre de paramètres dans le réseau et n le nombre de transitions.*

Un point $(\nu|\nu')$ du domaine de tir est alors constitué d'une valuation $\nu = (\lambda_1, \ldots, \lambda_l)$ des paramètres de Par et d'une valuation $\nu' = (\theta_1, \ldots, \theta_n)$ des dates de tir des transitions. L'ensemble de ces dates de tir sera noté Θ.

$D_{|Par}$ est la projection d'un domaine de tir D sur l'ensemble des paramètres (c'est-à-dire l'élimination des variables Θ des transitions) :

$$D_{|Par} = \{\nu \in \mathbb{Q}^{+l} \mid \exists \nu' \in \mathbb{R}^n \text{ t.q. } (\nu|\nu') \in D\}$$

4.5.2 Calcul du graphe des classes d'états paramétrées

Le graphe des classes d'états paramétrées est calculé de manière similaire au cas non paramétré. Les paramètres sont intégrés intialement dans le domaine de tir de la classe initiale. Les opérations du calcul des successeurs sont ensuite identiques et ne modifient pas directement les paramètres. Toutefois, le domaine des paramètres sera au fur et à mesure du calcul

automatiquement réduit afin de ne garder que les valuations des paramètres pour lesquelles la classe est accessible.

Définition 4.4 (Transition tirable depuis une classe d'états) *Soit une classe d'états paramétrée $C = (M, D)$ d'un PITPN. Une transition t_i est dite* tirable *depuis C si elle est sensibilisée et non inhibée par M, et s'il existe une solution $(\nu|\nu') \in D$, telle que $\forall j \in [1..n]$ t.q. $j \neq i$ et $t_j \in \mathsf{enabled}(M)$ et $t_j \notin \mathsf{inhibited}(M)$, $\nu'(\theta_i) \leq \nu'(\theta_j)$.*

Le calcul du successeur d'une classe paramétrée après le tir d'une transition est identique dans les trois premières étapes ; à la quatrième l'ajout de nouvelles inéquations pour les transitions nouvellement sensibilisées peut introduire des paramètres.

Définition 4.5 (Successeur d'une classe d'états paramétrée) *Soit une classe d'états paramétrée $C = (M, D)$ et une transition t_f tirable dans C. La classe d'états paramétrée $C' = (M', D')$, successeur de C par le tir de t_f, ce que l'on note $C' = \mathsf{succ}(C, t_f)$, est calculée de la manière suivante :*

- *$M' = M - {}^\bullet t_f + t_f^\bullet$*
- *D' est calculé à partir de D par les étapes suivantes, et noté $\mathsf{next}(D, t_f)$:*
  1. *intersection avec les contraintes de franchissabilité : $\forall j$ t.q. t_j est* active*, $\theta_f \leq \theta_j$*
  2. *substitution des variables pour toutes les transitions actives t_j : $\theta_j = \theta_f + \theta'_j$,*
  3. *élimination de toutes les variables relatives aux transitions désensibilisées par le tir de t_f,*
  4. *ajout des inéquations relatives aux transitions nouvellement sensibilisées :*

$$\forall t_k \in \uparrow \mathsf{enabled}(M, t_f),\ {}^\uparrow J_s(t_k) \leq \theta'_k \leq J_s(t_k)^\downarrow$$

Calcul pour un point Soit une classe d'états paramétrée $C = (M, D)$ d'un PITPN, pour un point $x = (\lambda_1, \ldots, \lambda_l, \theta_1, \ldots, \theta_n)$ du domaine de tir D (les λ_i sont les valeurs des paramètres, ils définissent une valuation ν, les θ_i sont les valeurs des dates de tir) on calcule formellement ses successeurs après le tir d'une transition t_f tirable depuis $(M, \{x\})$:

$$\mathsf{next}(\{x\}, t_f) = \left\{ (\lambda_1, \ldots, \lambda_l, \theta'_1, \ldots, \theta'_n),\ \forall i \in [1..n] \;\middle|\; \begin{array}{l} \theta'_i \in J_s(t_i)(\nu) \text{ si } \uparrow \mathsf{enabled}(t_i, M, t_f) \\ \theta'_i = \theta_i \text{ si } t_i \in \mathsf{enabled}(M) \\ \quad \text{et } t_i \in \mathsf{inhibited}(M) \\ \quad \text{et non } \uparrow \mathsf{enabled}(t_i, M, t_f) \\ \theta'_i = 0 \text{ si } t_i \notin \mathsf{enabled}(M) \\ \theta'_i = \theta_i - \theta_f \text{ sinon} \end{array} \right\}$$

Cet opérateur next est étendu à une union finie ou infinie de points pour définir formellement $\mathsf{next}(D, t_f)$. On note qu'il ne modifie pas les valeurs des paramètres.

Graphe des classes d'états paramétrées Le graphe des classes d'états paramétrées est alors généré itérativement en appliquant la fonction définissant les successeurs d'une classe d'états. Le domaine initial des paramètres D_p est ajouté en conjonction au domaine de tir de la classe initiale.

Étant donné un PITPN $\mathcal{N}$, on calcule le graphe des classes d'états paramétrées de $\mathcal{N}$ en construisant le système de transitions $\mathcal{G}(\mathcal{N}) = \langle \mathcal{C}, C_0, \twoheadrightarrow, \rangle$ tel que :

– $C_0 = (M_0, D_0)$ est la classe initiale : M_0 est le marquage initial de $\mathcal{N}$, D_0 est défini par $D_0 = D_p \ \wedge \ \{\forall t_k \in \mathsf{enabled}\,(M_0)\,,\ {}^{\uparrow}J_s(t_k) \leq \theta_k \leq J_s(t_k)^{\downarrow}\}$,
– $C \overset{t}{\twoheadrightarrow} C'$ ssi $\begin{cases} t \in \mathsf{firable}\,(C)\,, \\ C' = \mathsf{succ}(C,t), \end{cases}$
– $\mathcal{C} = \{C \mid C_0 \twoheadrightarrow^* C\}$, où $\twoheadrightarrow^*$ est la fermeture réflexive et transitive de $\twoheadrightarrow$.

Les relations d'égalité et d'inclusion de classes d'états sont identiques pour les classes d'états paramétrées. Le *graphe des classes d'états paramétrées* $\mathcal{G}'(\mathcal{N})$ d'un PITPN est défini comme le quotient de $\mathcal{G}$ par une relation d'équivalence $\mathcal{R}$ (soit l'égalité, soit l'inclusion) :

$$\mathcal{G}'(\mathcal{N}) = \mathcal{G}(\mathcal{N})/\mathcal{R}$$

Exemple 4.3 *On considère le PITPN de la figure 4.1. Le graphe des classes d'états paramétrées de ce système est présenté sur la figure 4.2. Dans les classes finales C_4, C_5, C_6 le domaine de tir ne contient plus que les paramètres a, b et c avec les contraintes du domaine initial D_p, plus d'éventuelles contraintes supplémentaires induites par les choix d'embranchements nécessaires pour arriver dans la classe finale. Ainsi, la classe C_4 est accessible à la condition que $2b \geq 5$, C_5 si $a + c \leq 5$ et C_6 si $a \geq 4$.*

Par rapport au graphe des classes présenté sur la figure 3.5, il est maintenant possible de tirer t_2 en C_3 et d'accéder ainsi au marquage de la place p_4, mais à la condition que $a + c \leq 5$. Ce n'était effectivement pas le cas pour l'exemple de la figure 3.5 où $a = 2$ et $c = 4$.

On note également que comparé au cas non paramétré, les 2 classes C_4 et C_6 sont différenciées. Le marquage final après l'entrelacement de t_3 et t_4 est pourtant le même, mais le choix de tirer une transition avant l'autre ou le contraire entraîne des contraintes différentes sur les paramètres.

Nous signalons enfin que les contraintes supplémentaires rajoutées au domaine des paramètres réduisent la taille du graphe des classes d'états paramétrées qui serait obtenu sinon, notamment en empêchant le tir de t_2 et t_3 dès la classe C_0.

4.5.3 Valuation du graphe des classes d'états paramétrées

À partir du graphe des classes d'états paramétrées d'un PITPN il est possible de choisir une valuation des paramètres et de remplacer dans les domaines de tir des classes d'états paramétrées tous les paramètres par leur valeur. On obtient alors un graphe non paramétré. Toutefois, certains domaines de tir peuvent par cette opération devenir vide, ce qui signifie dans ce cas que la classe d'états n'est pas accessible pour la valuation des paramètres choisie. Ces classes doivent donc être supprimées du graphe non paramétré obtenu. Nous montrons au final que ce graphe non paramétré correspond au graphe des classes d'états de l'ITPN obtenu pour la même valuation des paramètres.

Définition 4.6 (Valuation d'une classe d'états paramétrée) *Soit $C = (M, D)$ une classe d'états paramétrée d'un PITPN $\mathcal{N}$ et $\nu \in D_p$ une valuation des paramètres de $\mathcal{N}$. La valuation de C par ν est une classe d'états non paramétrée $[\![C]\!]_\nu = (M, [\![D]\!]_\nu)$ où*

$$[\![D]\!]_\nu = \{\nu' \in \mathbb{R}^n \mid (\nu|\nu') \in D\}$$

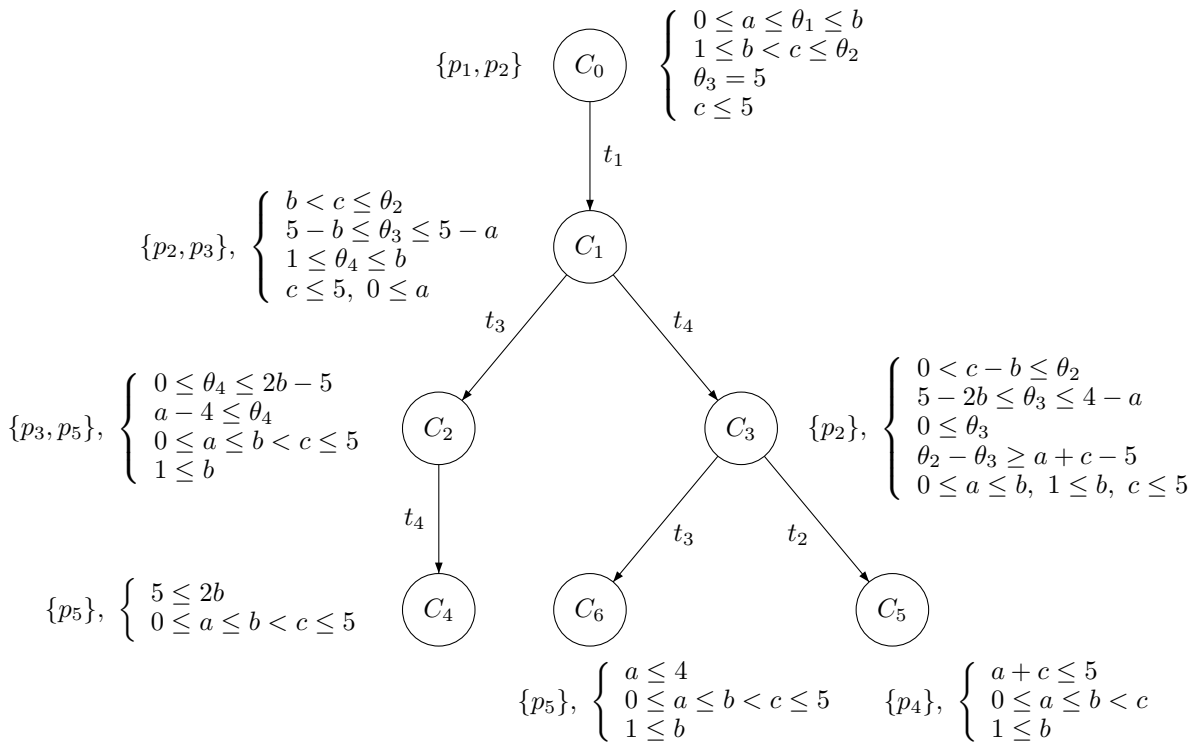


FIG. 4.2: Graphe des classes d'états paramétrées du PITPN de la figure 4.1

La valuation du graphe des classes d'états paramétrées notée $[\![\mathcal{G}(\mathcal{N})]\!]_\nu$ est obtenue en valuant les classes d'états paramétrées du graphe, en commençant par la classe initiale, et en supprimant les classes dont le domaine de tir devient vide, ainsi que leurs successeurs.

Définition 4.7 (Valuation du graphe des classes d'états paramétrées) *Étant donné un PITPN $\mathcal{N}$ et une valuation $\nu \in D_p$, la valuation du graphe des classes d'états paramétrées $\mathcal{G}(\mathcal{N})$, notée $[\![\mathcal{G}(\mathcal{N})]\!]_\nu$, est définie par le système de transitions $(\mathcal{C}_\nu, [\![C_0]\!]_\nu, \twoheadrightarrow)$, où :*

- *$[\![C_0]\!]_\nu$ est la valuation de la classe initiale C_0 de $\mathcal{G}(\mathcal{N})$;*
- *$[\![C]\!]_\nu \overset{t}{\twoheadrightarrow} [\![C']\!]_\nu$ ssi $\begin{cases} C = (M, D), C' = (M', D') \in \mathcal{G}(\mathcal{N}) \\ \text{et } C \overset{t}{\rightarrow} C' \\ \text{et } [\![D']\!]_\nu \neq \emptyset \end{cases}$*
- *$\mathcal{C}_\nu = \{[\![C]\!]_\nu \mid [\![C_0]\!]_\nu \twoheadrightarrow^* [\![C]\!]_\nu\}$, où $\twoheadrightarrow^*$ est la fermeture réflexive et transitive de $\twoheadrightarrow$.*

On peut appliquer à nouveau le quotient par la relation d'égalité entre classes non paramétrées, ce qui permet de regrouper des classes égales pour la valuation ν, qui ne l'étaient pas avec les paramètres.

Le théorème 4.2 établit que la valuation du graphe des classes d'états paramétrées d'un PITPN correspond au graphe des classes d'états non paramétré de l'ITPN obtenu pour la même valuation des paramètres.

Théorème 4.2 *Étant donné un PITPN $\mathcal{N}$ et une valuation $\nu \in D_p$, la relation d'égalité $\equiv$ entre classes est une relation de bisimulation entre $[\![\mathcal{G}(\mathcal{N})]\!]_\nu$ et $\mathcal{G}([\![\mathcal{N}]\!]_\nu)$:*

$$[\![\mathcal{G}(\mathcal{N})]\!]_\nu \equiv \mathcal{G}([\![\mathcal{N}]\!]_\nu)$$

Preuve 4.2 (Théorème 4.2) *Par définition* $[\![\mathcal{G}(\mathcal{N})]\!]_\nu = (\mathcal{C}_\nu, [\![C_0]\!]_\nu, \twoheadrightarrow)$ *et on note* $\mathcal{G}([\![\mathcal{N}]\!]_\nu) = (\mathcal{C}^*, C_0^*, \twoheadrightarrow)$.

1. *Nous montrons premièrement que* $[\![C_0]\!]_\nu \equiv C_0^*$ *:*
   D'une part, $[\![C_0]\!]_\nu = (M_0, [\![D_0]\!]_\nu)$ *où* $D_0 = D_p \wedge \{\theta_k \in J_s(t_k) \mid t_k \in \mathsf{enabled}\,(M_0)\}$*, et puisque* $\nu \in D_p$ *on obtient* $[\![D_0]\!]_\nu = \{\theta_k \in J_s(t_k)(\nu) \mid t_k \in \mathsf{enabled}\,(M_0)\}$.
   D'autre part, par définition, $C_0^* = (M_0, D_0^*)$ *où* $D_0^* = \{\theta_k \in J_s(t_k)(\nu) \mid t_k \in \mathsf{enabled}\,(M_0)\}$.
2. *Maintenant, soit* $[\![C]\!]_\nu \in [\![\mathcal{G}(\mathcal{N})]\!]_\nu$ *et* $C^* \in \mathcal{G}([\![\mathcal{N}]\!]_\nu)$ *telles que* $[\![C]\!]_\nu \equiv C^*$ *(nous notons alors* M *leur marquage), nous montrons que* $\forall t \in \mathsf{enabled}\,(M)$,
   $(\exists C' \text{ t.q. } [\![C]\!]_\nu \overset{t}{\twoheadrightarrow} [\![C']\!]_\nu) \iff (\exists C^{**} \text{ t.q. } C^* \overset{t}{\twoheadrightarrow} C^{**})$ *et* $[\![C']\!]_\nu \equiv C^{**}$ *:*

   (a) *Si* $[\![C]\!]_\nu \overset{t}{\twoheadrightarrow} [\![C']\!]_\nu$ *alors* $\begin{cases} C = (M, D), C' = (M', D') \in \mathcal{G}(\mathcal{N}) \\ C \overset{t}{\twoheadrightarrow} C' \\ [\![D']\!]_\nu \neq \emptyset \end{cases}$

   Puisque $[\![D']\!]_\nu \neq \emptyset$ *et* $C \overset{t}{\twoheadrightarrow} C'$*, cela signifie que* t *est tirable depuis* C *pour la valuation* ν*, i.e.* $\exists \nu'$*, t.q.* $(\nu|\nu') \in D$ *et* $\forall j \in [1..n]$*,* $j \neq i$*, t.q.* t_j *est active,* $\nu'(\theta_i) \leq \nu'(\theta_j)$*. Nous en déduisons alors que* $\nu' \in [\![D]\!]_\nu = D^*$*, et que* $t \in \mathsf{firable}\,(C^*)$*. Il existe donc* C^{**} *tel que* $C^* \overset{t}{\twoheadrightarrow} C^{**}$.

   (b) *Inversement, si* $C^* \overset{t}{\twoheadrightarrow} C^{**}$ *alors* $t \in \mathsf{firable}\,(C^*)$ *et donc* $\exists \nu' \in D^*$ *tel que* $\forall j \in [1..n]$*,* $j \neq i$*, t.q.* t_j *est active,* $\nu'(\theta_i) \leq \nu'(\theta_j)$*. Puisque* $\nu' \in D^* = [\![D]\!]_\nu$ *cela signifie que* $(\nu|\nu') \in D$ *et par conséquence nous en déduisons immédiatement que* $t \in \mathsf{firable}\,(C)$*. Il existe donc* C' *telle que* $C \overset{t}{\twoheadrightarrow} C'$ *et puisque* $t \in \mathsf{firable}\,(C)$ *pour le point* $(\nu|\nu')$ *cela signifie nécessairement que* $[\![D']\!]_\nu \neq \emptyset$.

   Finalement nous devons montrer que $C^{**} \equiv [\![C']\!]_\nu$*. Nous obtenons immédiatement l'égalité des marquages. Concernant les deux domaines de tir :*
   – *d'une part* $D' = \bigcup_{x \in D} \mathsf{next}(\{x\}, t)$ *et donc* $[\![D']\!]_\nu = \bigcup_{x \in D} \mathsf{next}(\{x\}, t)$ *tel que* $x = (\underbrace{\lambda_1, \ldots, \lambda_l}_{\nu}, \theta_1, \ldots, \theta_n)$ *;*
   – *d'autre part,* $D^{**} = \bigcup_{x \in D^*} \mathsf{next}(\{x\}, t)$.
   Et si $x = (\theta_1, \ldots, \theta_n) \in D^*$*, puisque* $D^* = [\![D]\!]_\nu$ *cela signifie que* $(\lambda_1, \ldots, \lambda_l, \theta_1, \ldots, \theta_n) \in D$ *(et réciproquement). En conséquence, puisque l'opérateur* next *est le même dans le cas paramétré et non paramétré (il ne modifie pas les paramètres), les deux domaines sont égaux.*

Finalement, le théorème 4.3 et le lemme 4.4 permettent de vérifier directement l'accessibilité d'une classe d'états paramétrée pour une valuation des paramètres, en calculant ce que nous nommons la *condition d'accessibilité* de la classe d'états paramétrée.

Définition 4.8 (Condition d'accessibilité) *Étant donnée une classe d'états paramétrée* $C = (M, D)$*,* $D_{|Par}$ *est la* condition d'accessibilité *de* C.

Théorème 4.3 *Étant donné un PITPN* $\mathcal{N}$ *et une valuation* $\nu \in D_p$*, soit* $C = (M, D)$ *une classe d'états paramétrée de* $\mathcal{G}(\mathcal{N})$*, alors :*

$$[\![C]\!]_\nu \in [\![\mathcal{G}(\mathcal{N})]\!]_\nu \text{ ssi } \nu \in D_{|Par}$$

Ce théorème est dû au fait que la condition d'accessibilité d'une classe, qui est un polyèdre dans l'espace des paramètres, contient toujours les conditions d'accessibilité de tous ses successeurs, ce qui est prouvé par le lemme 4.4.

Lemme 4.4 *Étant donné un PITPN $\mathcal{N}$, soit $C = (M, D)$ et $C' = (M', D')$ deux classes d'états paramétrées de $\mathcal{G}(\mathcal{N})$. Si $C \overset{t}{\twoheadrightarrow} C'$ alors $D'_{|Par} \subseteq D_{|Par}$.*

Preuve 4.3 (Lemme 4.4) *Soit $\nu \in D'_{|Par}$. Par définition de la projection il existe une valuation $\nu' \in \mathbb{R}^n$ des dates de tir de D' telle que $(\nu|\nu') \in D'$. $C' = \mathsf{succ}(C, t)$ et donc $D' = \mathsf{next}(D, t)$. $(\nu|\nu') \in D'$ entraîne alors qu'il existe une valuation $\nu'' \in \mathbb{R}^n$ des dates de tir de C telle que $(\nu|\nu'') \in D$ et $\mathsf{next}(\{(\nu|\nu'')\}, t) = \{(\nu|\nu')\}$ (puisque les paramètres ne sont pas modifiés par l'opérateur* next*). A nouveau, par définition de la projection cela entraîne que $\nu \in D_{|Par}$ ce qui prouve le lemme.*

Preuve 4.4 (Théorème 4.3) *Nous déduisons directement à partir de la définition de $[\![\mathcal{G}(\mathcal{N})]\!]_\nu$ que si $[\![C]\!]_\nu \in [\![\mathcal{G}(\mathcal{N})]\!]_\nu$ alors $[\![D]\!]_\nu \neq \emptyset \Rightarrow \exists \nu' \in \mathbb{R}^n$ t.q. $(\nu|\nu') \in D \Rightarrow \nu' \in D'_{|Par}$.*
Inversement, puisque $C \in \mathcal{G}(\mathcal{N})$ il existe nécessairement une séquence $C_0 \twoheadrightarrow C_1 \twoheadrightarrow \cdots \twoheadrightarrow C_{k-1} \twoheadrightarrow C_k = C$ avec $C_1, \ldots, C_k - 1 \in \mathcal{G}(\mathcal{N})$. Grâce au lemme 4.4 nous démontrons alors les inclusions suivantes : $D_{|Par} = D_{k|Par} \subseteq D_{k-1|Par} \subseteq \cdots \subseteq D_{1|Par} \subseteq D_{0|Par}$. Et donc si $\nu \in D_{|Par}$, alors $\forall 0 \leq i \leq k$, $\nu \in D_{i|Par}$, ce qui implique que $[\![D_i]\!]_\nu \neq \emptyset$. Avec ces conditions, en partant de la classe initiale C_0, on montre récursivement que $[\![C_i]\!]_\nu \in [\![\mathcal{G}(\mathcal{N})]\!]_\nu$, en particulier pour $i = k$: $[\![C]\!]_\nu \in [\![\mathcal{G}(\mathcal{N})]\!]_\nu$.

Exemple 4.4 *Si l'on choisit la valuation $\nu = (2, 3, 4)$ pour les paramètres (a, b, c), à partir du PITPN $\mathcal{N}$ de la figure 4.1 on obtient pour cette valuation l'ITPN $[\![\mathcal{N}]\!]_\nu$ de la figure 3.3, et si l'on considère le graphe des classes d'états paramétrées de $\mathcal{N}$ présenté sur la figure 4.2, en remplaçant dans ce graphe les paramètres par leur valeur on obtient alors logiquement, tel que prévu par le théorème 4.2, le graphe des classes d'états de $[\![\mathcal{N}]\!]_\nu$ présenté sur la figure 3.5 (pour obtenir un graphe isomorphe il faut regrouper les classes égales $[\![C_4]\!]_\nu$ et $[\![C_6]\!]_\nu$).*

4.6 Model-checking paramétré de formules TCTL paramétrées

Le problème du model-checking consiste à vérifier qu'un modèle $\mathcal{N}$ satisfait une propriété ϕ exprimée dans une logique donnée, ce que l'on écrit plus formellement $\mathcal{N} \models \phi$. La réponse à ce problème ne peut être que *vrai* ou *faux*.

Étant donné un modèle paramétré $\mathcal{N}$ et une propriété ϕ, qui peut également être paramétrée, nous souhaitons résoudre le *problème de synthèse des paramètres*, c'est-à-dire que nous voulons déterminer l'ensemble des valeurs des paramètres $\Gamma(\mathcal{N}, \phi)$ tel que $\forall \nu \in \Gamma(\mathcal{N}, \phi)$ le modèle non paramétré $[\![\mathcal{N}]\!]_\nu$ obtenu pour la valuation ν satisfait la propriété non paramétrée $[\![\phi]\!]_\nu$ obtenue pour la même valuation, ce que l'on écrit formellement $[\![\mathcal{N}]\!]_\nu \models [\![\phi]\!]_\nu$. Cet ensemble de solutions sera représenté par un système de contraintes sur les paramètres du problème.

4.6.1 Formules TCTL paramétrées

De la même manière que nous avons paramétré le modèle des réseaux de Petri avec arcs inhibiteurs, nous pouvons paramétrer les formules TPN-TCTL en autorisant l'utilisation de

paramètres dans les intervalles temporels des formules. Les paramètres utilisés dans les formules sont communs au PITPN étudié et sont donc dans l'ensemble Par des paramètres.

D'emblée nous nous limitons au sous-ensemble de formules TPN-TCTL$_\mathcal{S}$ pour lequel des algorithmes de model-checking à la volée ont été définis [Gardey 05a, Boucheneb 09, Hadjidj 06]. Ce sont ces méthodes que nous allons adapter au model-checking paramétré.

Définition 4.9 (PTCTL$_\mathcal{S}$) *La syntaxe des formules TCTL paramétrées du sous-ensemble que nous notons PTCTL$_\mathcal{S}$ est définie par la grammaire suivante :*

$$PTCTL_\mathcal{S} := \exists\varphi\ \mathcal{U}_J\ \psi \mid \forall\varphi\ \mathcal{U}_J\ \psi \mid \exists\Diamond_J\varphi \mid \forall\Diamond_J\varphi \mid \exists\Box_J\varphi \mid \forall\Box_J\varphi \mid \varphi \rightsquigarrow_{J_r} \psi$$

où $\varphi, \psi \in PR$ sont des propositions sur les marquages du réseau, $J, J_r \in \mathcal{J}(Par)$ sont des intervalles temporels paramétrés avec la restriction que $J_r = [0, b]$ où $b \in \mathbb{Q}^+ \cup Par$, ou $J_r = [0, \infty[$.

La sémantique des formules PTCTL$_\mathcal{S}$ sur un PITPN est définie de manière similaire à la sémantique des PITPN, c'est-à-dire en considérant une valuation des paramètres et en remplaçant les paramètres de la formule par leur valeur ; cette formule non paramétrée est alors interprétée sur le modèle ITPN non paramétré obtenu pour la même valuation.

Définition 4.10 (Sémantique de PTCTL$_\mathcal{S}$) *Étant donné un PITPN $\mathcal{N}$, une formule PTCTL$_\mathcal{S}$ ϕ, et une valuation $\nu \in D_p$ des paramètres de $\mathcal{N}$ (qui sont partagés avec ϕ), $[\![\phi]\!]_\nu$ est la formule TPN-TCTL obtenue en remplaçant dans ϕ l'intervalle paramétré J (ou J_r) par l'intervalle de $\mathbb{Q}^+$ $J(\nu)$ (ou $J_r(\nu)$).*
On dit alors que $\mathcal{N}$ satisfait ϕ pour la valuation ν si et seulement si $[\![\mathcal{N}]\!]_\nu \models [\![\phi]\!]_\nu$.

4.6.2 Ajout d'une horloge globale au graphe des classes d'états paramétrées

Dans le graphe des classes d'états, le domaine de tir d'une classe décrit le domaine des dates de tir des transitions sensibilisées. L'origine temporelle de ces dates est la date d'arrivée dans la classe, c'est-à-dire la date de tir de la transition tirée précédemment. Les propriétés temporelles sont difficiles à vérifier dans ce contexte.

Pour pouvoir vérifier des propriétés temporelles de type TCTL sur le graphe des classes il faut être capable d'évaluer le temps écoulé entre deux classes. Nous reprenons pour cela l'approche développée dans [Gardey 05a, Boucheneb 09, Penczek 04], qui consiste à rajouter une transition spéciale au réseau, chargée de mesurer le temps écoulé. Cette transition ne sera jamais tirée, la méthode se résume donc à rajouter une variable aux domaines de tir des classes que nous noterons θ_c. Cette variable est initialisée à zéro dans la classe initiale puis décroît avec l'évolution du temps, comme les autres variables de transitions. Contrairement aux autres variables sa valeur sera donc toujours négative [1]. Par contre elle ne doit pas contraindre les autres variables des transitions lors du calcul des contraintes de franchissabilité. Ceci étant, le temps écoulé depuis l'initialisation de θ_c jusqu'à l'entrée dans la classe sera donné par $\tau_c = -\theta_c$.

[1] Cela ne pose pas de problème dans la construction du graphe des classes. Une alternative serait de déterminer une valeur suffisamment large pour l'initialisation, mais cela semble plus compliqué.

Définition 4.11 (Classe d'états paramétrée étendue) *Une* classe d'états paramétrée étendue $C = (M, D)$ *d'un PITPN est une classe d'états dont le domaine de tir D comporte une variable supplémentaire $\theta_c \in \Theta$ qui n'est pas associée à une transition.*

La définition 4.4 de la franchissabilité d'une transition depuis une classe d'états étendue n'est pas modifiée. En effet, les contraintes de franchissabilité ne concernent que les variables θ_i telles que $\forall i \in [1..n]$, $t_i \in T$; cela n'inclut donc pas θ_c qui comme souhaité ne contraint alors pas les variables des transitions.

Le calcul des successeurs par l'opérateur next est lui redéfini de sorte que pour un point $x = (\lambda_1, \ldots, \lambda_l, \theta_1, \ldots, \theta_n, \theta_c)$ du domaine D d'une classe d'états étendue, les successeurs de x après le tir d'une transition t_f définis dans $\mathsf{next}((\{x\}, t_f)$ sont tels que $\theta'_c = \theta_c - \theta_f$.

Le graphe des classes d'états paramétrées étendues $\mathcal{G}_c(\mathcal{N})$ est alors calculé itérativement de manière similaire, en partant de la classe initiale $C_0 = (M_0, D_0)$ où :

$$D_0 = D_p \wedge \{\theta_k \in J(t_k) \mid t_k \in \mathsf{enabled}\,(M_0)\} \wedge \{\theta_c = 0\}$$

Calcul du temps écoulé À partir d'une classe d'états paramétrée étendue $C = (M, D)$ nous pouvons déterminer les fonctions suivantes :

- $\tau_{min}(C)$, est le temps minimum absolu (depuis l'initialisation de θ_c) écoulé à l'arrivée dans la classe C. C'est une fonction sur l'espace des paramètres : $\tau_{min}(C) : D_p \to \mathbb{Q}^+$, telle que :
$$\tau_{min}(C)(\nu) = \min_{x=(\nu|\nu')\in D}(\tau_c)$$
Elle peut être représentée comme le maximum entre les valeurs minimales de τ_c et est nécessairement positive et finie.
- $\tau_{max}(C)$, est le temps maximum absolu (depuis l'initialisation de θ_c) écoulé à l'arrivée dans la classe C. C'est une fonction sur l'espace des paramètres : $\tau_{max}(C) : D_p \to \mathbb{Q}^+ \cup \{\infty\}$, telle que :
$$\tau_{max}(C)(\nu) = \max_{x=(\nu|\nu')\in D}(\tau_c)$$
Elle peut être représentée comme le minimum entre les valeurs maximales de τ_c et est nécessairement positive mais peut être infinie s'il n'existe pas de temps maximum.

Proposition 4.5 *Étant donnée une classe d'états étendue $[\![C]\!]_\nu \in [\![\mathcal{G}_c(\mathcal{N})]\!]_\nu$, soit $q \in [\![C]\!]_\nu$ un état accessible par une exécution ρ. Alors le temps écoulé* $\mathsf{time}(\rho, q)$ *à l'état q depuis l'état initial q_0 est tel que :*

$$\tau_{min}(C)(\nu) \leq \mathsf{time}(\rho, q) \leq \tau_{max}(C)(\nu)$$

La proposition 4.5 est équivalente à dire qu'il existe un point x dans le domaine $[\![D]\!]_\nu$ de la classe étendue tel qu'en ce point $\tau_c = -\theta_c = \mathsf{time}(\rho, q)$.

Preuve 4.5 (Proposition 4.5) *q est un état accessible de $[\![C]\!]_\nu$ par une exécution $\rho = q_0 \xrightarrow{(d_0,t_0)} q_1 \xrightarrow{(d_1,t_1)} q_2 \cdots q_n \xrightarrow{(d_n,t_n)} q$ de $[\![\mathcal{N}]\!]_\nu$, et par définition $\mathsf{time}(\rho, q) = d_0 + d_1 + \cdots + d_n$. Par la séquence de transitions $s = t_0; t_1; \ldots; t_n$ le chemin suivant dans $[\![\mathcal{G}_c(\mathcal{N})]\!]_\nu$: $[\![C_0]\!]_\nu \xrightarrow{t_0} [\![C_1]\!]_\nu \xrightarrow{t_1} \cdots [\![C_n]\!]_\nu \xrightarrow{t_n} [\![C]\!]_\nu$ est également accessible. Par la définition du calcul des successeurs, la variable θ_c est initialisée à 0 dans C_0, et est décrémentée des dates de tirs successives des transitions i.e. $\theta_c = 0 - \theta(t_0) - \theta(t_1) - \cdots - \theta(t_n)$ (où pour i de 0 à n, $\theta(t_i)$ est la date de tir de la transition t_i dans la classe C_i). Or puisque t_0 est tirable dans l'exécution ρ après*

d_0 unités de temps, cela signifie qu'il existe dans C_0 une valeur $\theta(t_0) = d_0$. Il en est de même pour tout i de 0 à n. Donc il existe dans $[\![D]\!]_\nu$ une valeur $\theta_c = -d_0 - d_1 - \cdots - d_n$.
En conclusion $\exists \tau_c$, $\mathsf{time}(\rho, q) = -\theta(c) = \tau_c$, ce qui prouve par définition des fonctions $\tau_{min}(C)$ et $\tau_{max}(C)$ que $\tau_{min}(C)(\nu) \leq \mathsf{time}(\rho, q) \leq \tau_{max}(C)(\nu)$.

Lorsque l'on choisit un état q dans une classe d'états étendue, il est accessible par une exécution ρ. Suivant l'éxécution choisie, le temps écoulé $\mathsf{time}(\rho, q)$ peut être différent. La proposition 4.5 montre que le choix d'un état et d'une exécution y amenant, se ramène à choisir une valeur θ_c dans le domaine de tir de la classe étendue. Dorénavant, lors du choix d'un état dans une classe d'états étendue, on déterminera également une valeur de θ_c, ce qui définit le temps écoulé depuis l'initialisation de θ_c jusqu'à l'état q, que nous noterons simplement $\mathsf{time}(q)$

Exemple 4.5 *Considérons à nouveau le PITPN de la figure 4.1, son graphe des classes d'états paramétrées étendues est maintenant donné dans la figure 4.3. On peut par exemple s'intéresser au temps écoulé à l'arrivée dans la classe C_4. En regardant les contraintes sur θ_c, on calcule ainsi que :*

- *$\tau_{min}(C_4) = \max(5, a+1)$. En effet, pour arriver en C_4 on a du tirer t_3, donc attendre au minimum 5 unités de temps, et on a tiré en parallèle t_1 et t_4 donc un temps d'exécution minimum de $a + 1$.*
- *$\tau_{max}(C_4) = 2b$. Cela correspond à la somme des temps maximum d'exécution de t_1 et t_4, sachant qu'en C_4, $2b \geq 5$, car t_4 est tirée après t_3.*

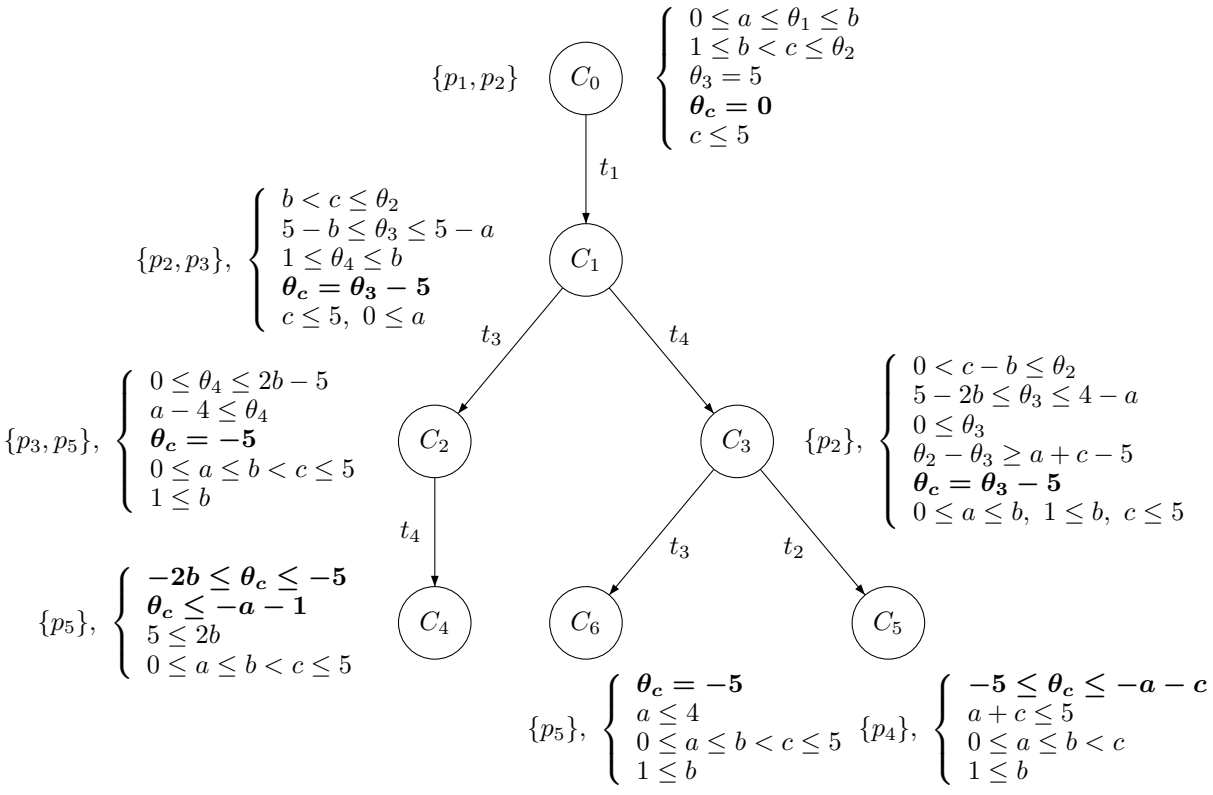


FIG. 4.3: Graphe des classes d'états paramétrées étendues du PITPN de la figure 4.1

4.6.3 Principes du model-checking paramétré

Étant donné un PITPN $\mathcal{N}$ et une propriété PTCTL ϕ, nous voulons caractériser l'ensemble $\Gamma(\mathcal{N}, \phi)$ de toutes les valuations des paramètres qui permettent de résoudre le problème de model-checking, c'est-à-dire :

$$\Gamma(\mathcal{N}, \phi) = \{\nu \in D_p \mid [\![\mathcal{N}]\!]_\nu \models [\![\phi]\!]_\nu\}$$

Pour cela la solution que nous proposons est de calculer récursivement, sur chaque classe d'états paramétrée étendue $C = (M, D)$, un prédicat logique sur les paramètres qui correspond à la vérification de la propriété sur la classe courante et sur ses successeurs. Ce prédicat représentera l'ensemble suivant :

$$F^\phi(C) = \{\nu \in D_{|Par} \mid [\![C]\!]_\nu \models [\![\phi]\!]_\nu\}$$

Nous commençons par définir une interprétation de la vérification d'une formule PTCTL ϕ sur une classe d'états paramétrée étendue $C = (M, D)$, ce que nous écrivons $[\![C]\!]_\nu \models [\![\phi]\!]_\nu$.

Interprétation de $[\![C]\!]_\nu \models [\![\phi]\!]_\nu$ pour $\phi = \exists\varphi\ \mathcal{U}_J\ \psi$ ou $\phi = \forall\varphi\ \mathcal{U}_J\ \psi$

Pour une valuation $\nu \in D_{Par}$ et un état $q \in [\![C]\!]_\nu$, on définit $[\![\phi[J - \mathsf{time}(q)]]\!]_\nu$ comme la formule TCTL obtenue en remplaçant dans ϕ l'intervalle temporel paramétré J par $J(\nu) - \mathsf{time}(q)$.

Nous définissons alors suivant la forme de la formule PTCTL ϕ que :
- si $\phi = \exists\varphi\ \mathcal{U}_J\ \psi$, alors $[\![C]\!]_\nu \models [\![\phi]\!]_\nu$ ssi $\exists q \in [\![C]\!]_\nu, q \models [\![\phi[J - \mathsf{time}(q)]]\!]_\nu$
- si $\phi = \forall\varphi\ \mathcal{U}_J\ \psi$, alors $[\![C]\!]_\nu \models [\![\phi]\!]_\nu$ ssi $\forall q \in [\![C]\!]_\nu, q \models [\![\phi[J - \mathsf{time}(q)]]\!]_\nu$

Interprétation de $[\![C]\!]_\nu \models [\![\phi]\!]_\nu$ pour $\phi = \varphi \rightsquigarrow_{J_r} \psi$

Dans le sous-ensemble de formules PTCTL$_\mathcal{S}$, les formules utilisant l'opérateur $\rightsquigarrow$ de réponse bornée sont les seules à intégrer un niveau de récursivité dans la formule (par définition de $\rightsquigarrow$). Or ce sous-ensemble a été introduit pour proposer des algorithmes de vérification « à la volée » en profitant justement du fait que sans récursivité les formules peuvent être vérifiées par l'analyse directe du graphe des classes d'états.

Afin d'appliquer ces méthodes sur les formules utilisant la réponse bornée, on étend le modèle en ajoutant une place supplémentaire nommée P_{LT} au réseau de Petri. Elle sera marquée quand on cherche à vérifier ψ lors de la vérification de $\varphi \rightsquigarrow_{J_r} \psi$. On note alors $\mathcal{N}_{LT}$ le modèle PITPN étendu obtenu.

Dans ce modèle le calcul des successeurs d'une classe d'états est modifié afin de mettre à jour le marquage de la place P_{LT}. Ainsi, le successeur $C' = (M', D') = succ_{LT}(C, t)$ d'une classe d'états paramétrée étendue $C = (M, D) \in \mathcal{G}_c(\mathcal{N}_{LT})$ par le tir d'une transition $t_f \in \mathsf{firable}(C)$, est donné par :

- $M' = M - {}^\bullet t_f + t_f^\bullet$ et $\begin{cases} \text{si } (M' \models \varphi \text{ et } M' \not\models \psi) \text{ alors } M'(P_{LT}) = 1, \\ \text{sinon si } (M \models \psi) \text{ alors } M'(P_{LT}) = 0, \\ \text{sinon } M'(P_{LT}) = M(P_{LT}) \end{cases}$
- $D' = \mathsf{next}(D, t_f)$ et
  si $(M(P_{LT}) = 0$ ou $M \models \psi)$ alors la variable θ_c est réinitialisée à zéro.

On définit alors sur ce modèle que $[\![C]\!]_\nu \models [\![\phi]\!]_\nu$ ssi $\forall q \in [\![C]\!]_\nu$, $\forall \rho \in \pi(q)$,

$$M(P_{LT}) = 1 \Rightarrow \begin{cases} \exists 0 \leq r_1 \leq J_r(\nu)^\downarrow - \mathsf{time}(q) \text{ t.q. } \rho^*(r_1) \models \psi \text{ et} \\ \forall r_2 \geq r_1 \ \rho^*(r_2) \models M(P_{LT}) = 1 \Rightarrow \exists r_3 \geq r_2 \\ \qquad\qquad \text{t.q. } r_3 - r_2 \leq J_r(\nu)^\downarrow \text{ et } \rho^*(r_3) \models \psi \end{cases}$$

$$M(P_{LT}) = 0 \Rightarrow \begin{cases} \forall r_2 \geq 0 \ \rho^*(r_2) \models M(P_{LT}) = 1 \Rightarrow \exists r_3 \geq r_2 \\ \qquad\qquad \text{t.q. } r_3 - r_2 \leq J_r(\nu)^\downarrow \text{ et } \rho^*(r_3) \models \psi \end{cases}$$

$\mathsf{time}(q)$ fait ici référence au temps écoulé depuis la dernière réinitialisation de l'horloge θ_c. Nous remarquons que lorsque l'horloge vient d'être réinitialisée (donc $\mathsf{time}(q) = 0$) alors les deux définitions ci-dessus sont équivalentes et correspondent à $q \models [\![\phi]\!]_\nu$.

Finalement, le théorème 4.6 établit que nous sommes capables de résoudre le problème du model-checking paramétré en calculant l'ensemble des solutions sur la classe initiale.

Théorème 4.6 *Étant donné un PITPN $\mathcal{N}$ et une formule PTCTL$_\mathcal{S}$ ϕ,*

$$\Gamma(\mathcal{N}, \phi) = F^\phi(C_0)$$

où C_0 est la classe d'états initiale du graphe des classes d'états paramétrées étendues de $\mathcal{N}$.

Preuve 4.6 (Théorème 4.6) *À partir de leur définition respective on a :*

D'une part, $\Gamma(\mathcal{N}, \phi) = \{\nu \in D_p \mid [\![\mathcal{N}]\!]_\nu \models [\![\phi]\!]_\nu\}$ et par définition : $[\![\mathcal{N}]\!]_\nu \models [\![\phi]\!]_\nu \Leftrightarrow q_0 \models [\![\phi]\!]_\nu$, où q_0 est l'état initiale de $[\![\mathcal{N}]\!]_\nu$.

D'autre part, pour $C_0 = (M_0, D_0)$:

- *si $\phi = \exists \varphi \, \mathcal{U}_J \, \psi$ ou $\phi = \forall \varphi \, \mathcal{U}_J \, \psi$, alors $F^\phi(C_0) = \{\nu \in D_{0|Par} \mid q_0 \models [\![\phi[J - \mathsf{time}(q_0)]]\!]_\nu\}$, car la classe initiale $[\![C_0]\!]_\nu$ se confond avec l'état initial $q_0 = (M_0, J_s(\nu))$. De plus, $D_{0|Par} = D_p$ et $\mathsf{time}(q_0) = 0$ (donc $\phi[J - \mathsf{time}(q_0)] = \phi$).*
- *si $\phi = \varphi \rightsquigarrow_{J_r} \psi$ alors de la même manière $[\![C_0]\!]_\nu$ est confondue avec l'état initial q_0 et $\mathsf{time}(q_0) = 0$. En conséquence, la définition de $[\![C_0]\!]_\nu$ donnée précédemment correspond à celle de $q_0 \models [\![\phi]\!]_\nu$.*

En conclusion on s'aperçoit syntaxiquement que quelle que soit la forme de la formule ϕ, les deux ensembles sont égaux.

4.6.4 Semi-algorithmes de model-checking paramétré

Pour vérifier des formules PTCTL$_\mathcal{S}$ nous proposons alors trois semi-algorithmes (puisque le problème est indécidable) qui dépendent de la forme de la formule considérée. Ils suivent le même principe qui est de caractériser récursivement pour chaque classe d'états C l'ensemble $F^\phi(C)$. Cet ensemble sera représenté par des conjonctions ou disjonctions de contraintes linéaires sur les paramètres. On utilisera en pratique une forme normale disjonctive (*i.e.* une disjonction de polyèdres). Ces trois semi-algorithmes sont suffisants pour vérifier toutes les formes de formules PTCTL$_\mathcal{S}$. Notamment les autres formes $\exists \Box_J \varphi$ et $\forall \Box_J \varphi$ sont vérifiées en calculant la négation du résultat respectivement de $\forall \Diamond_J \neg\varphi$ et $\exists \Diamond_J \neg\varphi$.

Dans ces semi-algorithmes, nous utilisons les valeurs $\tau_{max}(C)$ et $J^\downarrow$ (respectivement le temps maximum écoulé dans la classe C et la borne maximum de la formule PTCTL), or elles peuvent être infinies. En conséquence, dans les contraintes utilisant ces valeurs, les opérateurs

de comparaison $\leq$ et $\geq$ sont étendus à $\mathbb{R}^+ \cup \{\infty\}$. En particulier, si $\tau_{max}(C) = \infty$ et $J^{\downarrow} = \infty$, la contrainte $\tau_{max}(C) \leq J^{\downarrow}$ sera considérée vraie par convention.

Les preuves de la correction et de la complétude de ces semi-algorithmes sont données en annexe A.

4.6.4.1 Semi-algorithme EU

Ce semi-algorithme est conçu pour les formules PTCTL$_{\mathcal{S}}$ dont la forme est $\phi = \exists \varphi \, \mathcal{U}_J \, \psi$, où $J \in \mathcal{J}(Par)$ et $\phi, \psi \in PR$

Soit une classe d'états paramétrée étendue $C = (M, D) \in \mathcal{G}_c(\mathcal{N})$, on calcule :

$$F^{\phi}_{\mathrm{EU}}(C) = D_{|Par} \ \wedge \ \{\tau_{min}(C) \leq J^{\downarrow}\} \ \wedge \ \Bigg(\Big(M \models \psi \ \wedge \ \{\tau_{max}(C) \geq {}^{\uparrow}J\} \Big)$$
$$\vee \Big(M \models \varphi \ \wedge \ M \models \psi \ \wedge \ \big(\mathsf{firable}\,(C) = \emptyset \vee$$
$$\Big(\bigvee_{\substack{t \in \mathsf{firable}(C) \\ C' = (M', D') = \mathsf{succ}(C,t)}} (\{\tau_{max}(C') \geq {}^{\uparrow}J\} \wedge \ D'_{|Par})) \Big) \Big)$$
$$\vee \Big(M \models \varphi \ \wedge \ \mathsf{firable}\,(C) \neq \emptyset \ \wedge \ \Big(\bigvee_{\substack{t \in \mathsf{firable}(C) \\ C' = \mathsf{succ}(C,t)}} F^{\phi}_{\mathrm{EU}}(C') \Big) \Big) \Bigg)$$

Les deux premières conditions $D_{|Par} \wedge \{\tau_{min}(C) \leq J^{\downarrow}\}$ assurent que la classe est accessible et que le temps écoulé n'a pas dépassé la borne maximum de la formule ϕ. Ensuite trois conditions sont ajoutées en disjonction pour prouver ϕ :

- La première disjonction est utilisée lorsque C vérifie ψ mais pas ϕ. Dans ce cas pour vérifier ϕ il est nécessaire que le temps écoulé à l'entrée dans la classe soit dans l'intervalle temporel J de la formule ϕ, ce qui induit des contraintes sur les paramètres.
- La seconde disjonction est utilisée lorsque les conditions ϕ et ψ sont toutes deux vérifiées par C. Cela inclut le domaine de la première disjonction, mais de plus on peut maintenant attendre dans la classe afin que le temps écoulé soit compris entre les bonnes bornes temporelles. Les contraintes induites sont donc moins restrictives.
- La troisième disjonction est utilisée à chaque fois que C vérifie ϕ. On calcule alors les successeurs de C sur lesquels on itère l'algorithme. On ajoute les conditions $F^{\phi}_{\mathrm{EU}}(C')$ en disjonction.

Exemple 4.6 *Sur le PITPN de la figure 4.1 on peut vérifier par la formule PTCTL$_{\mathcal{S}}$* $\phi_1 = \exists \Diamond_{[d,\infty[}(M(p_3) > 0)$ *que la place p_3 peut être marquée après au minimum d unités de temps, où d est un paramètre supplémentaire introduit dans la formule. Le résultat du model-checking paramétré pour cette formule est donné ci-dessous (nous n'écrivons que les contraintes supplémentaires qui ne sont pas incluses dans le domaine initial des paramètres) :*

$$F^{\phi_1}_{EU}(C_0) = \{2b \geq d\}$$

4.6.4.2 Semi-algorithme AU

Ce semi-algorithme est conçu pour les formules PTCTL$_{\mathcal{S}}$ dont la forme est $\phi = \forall \varphi \, \mathcal{U}_J \, \psi$, où $J \in \mathcal{J}(Par)$ et $\phi, \psi \in PR$

Soit une classe d'états paramétrée étendue $C = (M, D) \in \mathcal{G}_c(\mathcal{N})$, on calcule :

$$F^{\phi}_{\mathrm{AU}}(C) = D_{|Par} \ \wedge \ \{\tau_{max}(C) \leq J^{\downarrow}\} \wedge \ \Big(\Big(M \models \psi \ \wedge \ \{\tau_{min}(C) \geq {}^{\uparrow}J\}\Big)$$
$$\vee \Big(M \models \varphi \ \wedge \ M \models \psi \ \wedge \ \Big(\mathsf{firable}\,(C) = \emptyset \vee$$
$$\Big(\bigwedge_{\substack{t \in \mathsf{firable}(C) \\ C'=(M',D')=\mathsf{succ}(C,t) \\ D''=D' \wedge \{\theta_c > -{}^{\uparrow}J\}}} (F^{\phi}_{\mathrm{AU}}(M', D'') \ \vee \ \neg D''_{|Par}))\Big)\Big)$$
$$\vee \Big(M \models \varphi \ \wedge \ \mathsf{firable}\,(C) \neq \emptyset \ \wedge \ \Big(\bigwedge_{\substack{t \in \mathsf{firable}(C) \\ C'=(M',D')=\mathsf{succ}(C,t)}} \big(F^{\phi}_{\mathrm{AU}}(C') \ \vee \ \neg D'_{|Par}\big)\Big)\Big)\Big)$$

L'algorithme utilise des conditions similaires au précédent. Cependant les contraintes ajoutées sont plus strictes à cause de l'utilisation de l'opérateur universel $\forall$.

- Ainsi, pour vérifier que le temps écoulé n'a pas dépassé la borne maximum de la formule, la contrainte utilise maintenant le temps maximum écoulé $\tau_{max}(C)$ au lieu du temps minimum.
- À l'inverse, dans la première disjonction on utilise maintenant le temps minimum écoulé au lieu du temps maximum.
- Dans la deuxième disjonction, il n'est plus suffisant de pouvoir attendre dans la classe pour certains points du domaine ; pour les points qui ne peuvent pas vérifier la condition de temps en restant dans la classe C, il est nécessaire de calculer leur successeurs et d'itérer l'algorithme.
- Enfin dans la troisième disjonction, on ajoute maintenant le résultat du calcul sur les successeurs dans une conjonction. En conséquence, pour chaque successeur on doit ajouter une disjonction de deux conditions : la condition calculée par l'itération de l'algorithme qui donne les valeurs des paramètres pour lesquelles la formule a été vérifiée, et au contraire la négation de la condition d'accessibilité du successeur, ce qui donne les valeurs des paramètres pour lesquelles le successeur n'est pas accessible.

Exemple 4.7 *Sur le PITPN de la figure 4.1, on peut vérifier la formule* $\phi_2 = \forall\Diamond_{[d,\infty[}(M(p_3) > 0)$ *qui impose contrairement à la formule* ϕ_1 *précédente que la place* p_3 *soit toujours marquée dans l'intervalle temporel paramétré* $[d, \infty[$. *Le résultat du model-checking paramétré pour* ϕ_2 *est cette fois :*

$$F^{\phi_2}_{AU}(C_0) = \{a \geq d - 1\}$$

4.6.4.3 Semi-algorithme LT

Ce semi-algorithme est conçu pour les formules PTCTL$_{\mathcal{S}}$ dont la forme est $\phi = \varphi \rightsquigarrow_{J_r} \psi$, où $J_r \in \mathcal{J}(Par)$ tel que $J_r = [0, b]$ avec $b \in \mathbb{Q}^+ \cup Par$ ou bien $J_r = [0, \infty[$, et $\phi, \psi \in PR$. Le modèle utilisé est ici $\mathcal{N}_{LT}$.

Soit une classe d'états paramétrée étendue $C = (M, D) \in \mathcal{G}_c(\mathcal{N}_{LT})$, on calcule :

$$F^{\phi}_{\mathrm{LT}}(C) = D_{|Par} \ \wedge \ \Big(M(P_{LT}) = 0 \ \vee \ \{\tau_{max}(C) \leq J_r^{\downarrow}\}\Big)$$
$$\wedge \Big(\Big(\mathsf{firable}\,(C) = \emptyset \ \wedge \ \big(M(P_{LT}) = 0 \ \vee \ M \models \psi\big)\Big) \vee$$
$$\Big(\mathsf{firable}\,(C) \neq \emptyset \ \wedge \ \big(\bigwedge_{\substack{t \in \mathsf{firable}(C) \\ C' = (M', D') = succ_{LT}(C,t)}} (F^{\phi}_{\mathrm{LT}}(C') \ \vee \ \neg D'_{|Par})\big)\Big)\Big)$$

L'algorithme est très similaire à l'algorithme AU, lorsque la borne minimale ${}^{\uparrow}J$ est nulle. La différence est qu'à cause de la récursivité de la formule l'analyse ne peut s'arrêter que si aucun successeurs n'existe.

Exemple 4.8 *Sur le PITPN de la figure 4.1 on vérifie la formule PTCTL suivante* $\phi_3 = (M(p_3) > 0) \rightsquigarrow_{[0,d]} (M(p_5) > 0)$ *qui spécifie que le tir de* t_1 *doit être suivi d'un tir de* t_5 *en moins de d unités de temps. Le résultat du model-checking paramétré est :*

$$F^{\phi_3}_{LT}(C_0) = \left\{ \begin{array}{l} a + d \geq 5 \\ a + c > 5 \end{array} \right\}$$

4.6.4.4 Convergence par inclusion

Afin de préserver les propriétés temporelles LTL, le graphe des classes d'états est défini avec un critère de convergence par égalité de classes d'états. Pour le model-checking de formules TCTL une fois que l'on a ajouté la variable θ_c aux domaines de tir la vérification de la formule ne dépend plus que du marquage et du domaine de tir de la classe courante. On va donc pouvoir utiliser un critère de convergence par inclusion de classes qui préserve l'accessibilité des marquages.

Pour une classe $C'' = (M'', D'')$ incluse dans une classe $C' = (M', D')$ on montre que :

1. $F^{\phi}_{\mathrm{EU}}(C'') \subseteq F^{\phi}_{\mathrm{EU}}(C')$: *En effet, puisque l'on a* $D''_{|Par} \subseteq D'_{|Par}$, $\tau_{min}(C'') \geq \tau_{min}(C')$, $\tau_{max}(C'') \leq \tau_{max}(C')$, *les conditions dans* $F^{\phi}_{EU}(C')$ *sont donc nécessairement plus larges ; de plus les successeurs de* C'' *sont nécessairement inclus dans ceux de* C', *ce qui permet de prouver par induction le résultat.*
2. À l'inverse, on peut montrer que si $C'' \subseteq C'$ alors $F^{\phi}_{\mathrm{AU}}(C') \subseteq (F^{\phi}_{\mathrm{AU}}(C'') \vee \neg D''_{|Par})$ ou encore $(F^{\phi}_{\mathrm{AU}}(C') \vee \neg D'_{|Par}) \subseteq (F^{\phi}_{\mathrm{AU}}(C'') \vee \neg D''_{|Par})$: *En effet, immédiatement si* $\nu \in \neg D'_{|Par}$ *alors* $\nu \in \neg D''_{|Par}$. *Considérons ensuite une valuation* $\nu \in F^{\phi}_{AU}(C')$. *Si* $\nu \notin D''_{|Par}$ *alors immédiatement on obtient* $\nu \in \neg D''_{|Par}$. *Sinon, on cherche à vérifier* $\nu \in F^{\phi}_{AU}(C'')$. *On obtient de nouveau* $\tau_{max}(C'') \leq \tau_{max}(C') \leq J^{\downarrow}$ *et si* $\tau_{min}(C') \geq J^{\downarrow}(\nu)$ *alors* $\tau_{min}(C'')(\nu) \geq \tau_{min}(C')(\nu) \geq J^{\downarrow}(\nu)$ *ce qui permet de vérifier la première disjonction. Dans les deux autres cas, on constate à nouveau que* $\forall t \in \mathsf{firable}\,(C'')$, $t \in \mathsf{firable}\,(C')$ *et* $succ(C'', t) \subseteq \mathsf{succ}(C', t)$. *Notons* $succ(C', t) = (M^*, D^*)$ *et* $succ(C'', t) = (M^{**}, D^{**})$. *Par hypothèse soit* $\nu \in \neg D^*_{|Par}$ *et donc nécessairement* $\nu \in \neg D^{**}_{|Par}$, *soit* $\nu \in F^{\phi}_{AU}(\mathsf{succ}(C', t))$ *et l'on peut à nouveau induire le résultat.*
3. On montre de la même manière que si $C'' \subseteq C'$ alors $(F^{\phi}_{\mathrm{LT}}(C') \vee \neg D'_{|Par}) \subseteq (F^{\phi}_{\mathrm{LT}}(C'') \vee \neg D''_{|Par})$.

Nous pouvons alors appliquer un critère de convergence par inclusion de classes d'états paramétrées. Dans le cas où une classe C'' est incluse dans une classe C', mais telle que C'' n'est pas un successeur de C' (*i.e.* $\neg(C \twoheadrightarrow^* C')$), donc $\exists C,\ C \twoheadrightarrow^* C'$ et $C \twoheadrightarrow^* C''$:

1. Dans le semi-algorithme EU, puisqu'au final les conditions sur chaque classe sont ajoutées en disjonction, à partir de la classe C on doit calculer : $F^{\phi}_{\mathrm{EU}}(C'') \vee F^{\phi}_{\mathrm{EU}}(C')$ ce qui est alors égal à $F^{\phi}_{\mathrm{EU}}(C')$; il n'est donc pas nécessaire de calculer $F^{\phi}_{\mathrm{EU}}(C'')$.
2. Dans le semi-algorithme AU, les résultats des classes sont au contraire ajoutés en conjonction, donc si $C'' \subseteq C'$ alors $(F^{\phi}_{\mathrm{AU}}(C') \vee \neg D'_{|Par}) \wedge (F^{\phi}_{\mathrm{AU}}(C'') \vee \neg D''_{|Par}) = F^{\phi}_{\mathrm{AU}}(C') \vee \neg D'_{|Par}$; donc dans ce cas également il n'est pas nécessaire de calculer $F^{\phi}_{\mathrm{AU}}(C'')$. Il en est de même pour le semi-algorithme LT.

4.7 Conclusion

Nous avons présenté dans ce chapitre un nouveau formalisme paramétré en ajoutant des paramètres temporels aux réseaux de Petri avec arcs inhibiteurs. Nous avons défini l'espace d'états symbolique de ce modèle paramétré à l'aide du graphe des classes d'états paramétrées. Cela nous a permis de proposer des semi-algorithmes de model-checking paramétré pour vérifier des formules TCTL paramétrées.

Dans le chapitre suivant nous allons décrire l'implémentation de ces fonctionnalités dans le logiciel Romeo en étudiant les différentes utilisations qui peuvent en être faites.

Chapitre 5

Implémentation et étude de cas

Résumé : Romeo *est un logiciel développé à l'IRCCyN dédié à la modélisation et à l'analyse des extensions temporelles des réseaux de Petri. Il réalise l'édition de réseaux de Petri, le calcul de l'espace d'états et la vérification de formules TCTL. Nous présentons dans ce chapitre l'implémentation des méthodes d'analyses des réseaux de Petri temporels paramétrés présentées au chapitre précédent. Nous appliquons ces fonctionnalités sur une étude de cas.*

La dernière version de Romeo *qui inclut les nouvelles fonctionnalités d'analyse des réseaux paramétrés a été présentée à la conférence TACAS (Tools and Algorithms for the Construction and Analysis of Systems) en 2009 [Lime 09].*

Sommaire

5.1 Présentation du logiciel Romeo

Le logiciel Romeo [Gardey 05b, Lime 09] est développé à l'IRCCyN au sein de l'équipe « Systèmes Temps Réel » sous la direction d'Olivier (H.) Roux. Il est composé d'une interface graphique (écrite en Tcl/Tk) permettant l'édition et la simulation de réseaux de Petri temporels et à chronomètres, éventuellement paramétrés, et d'un module de calcul Mercutio (écrit en C++) qui réalise des analyses d'espaces d'états et du model-checking TCTL. Il est distribué librement sous la licence CeCILL à l'adresse suivante : http://romeo.rts-software.org/, et est disponible pour les plateformes Windows, MacOSX et Linux.

Modélisation et simulation

L'interface graphique propose le choix entre trois modes temporels suivant l'extension désirée : les TPN, et deux extensions à chronomètres : les réseaux de Petri étendus à l'ordonnancement (Scheduling-TPN) [Bucci 04a, Roux 02] et les réseaux de Petri temporels avec arcs inhibiteurs [Roux 04]. Pour chacune de ces extensions, Romeo propose une extension paramétrée correspondante (PTPN, Scheduling-PTPN ou PITPN). Dans les modèles paramétrés Romeo supporte l'utilisation d'expression linéaires paramétrées dans les bornes temporelles des transitions et permet l'ajout de contraintes supplémentaires sur les paramètres. Une vue de l'interface graphique de Romeo pour l'édition de PITPN est présentée dans l'étude de cas à la fin de ce chapitre sur la figure 5.2.

L'interface graphique fournit également un simulateur interactif pour tester des scénarios. Il permet d'étudier une trace particulière dans l'espace d'états du modèle afin de détecter en première analyse des erreurs de modélisation. Plusieurs méthodes d'exploration de l'espace d'états sont pour cela disponibles : le graphe de classes d'états pour les TPN et les modèles à chronomètres, le graphe des zones pour les TPN seulement, et le graphe des classes d'états paramétrées pour les modèles paramétrés.

Calculs d'espaces d'états

Romeo implémente plusieurs méthodes de calcul de l'espace d'états des extensions temporelles des réseaux de Petri. Le graphe des classes d'états est utilisé dans les TPN et les SwPN ; il préserve les propriétés LTL [Berthomieu 91]. L'algorithme utilise des DBM pour les TPN bornés. Pour les SwPN le calcul exact n'est pas décidable mais peut être entrepris soit avec uniquement des polyèdres [Roux 04], soit de manière optimisée avec à la fois des DBM et des polyèdres [Magnin 05]. Pour les réseaux à chronomètres, Romeo propose également un semi-algorithme de surapproximation de l'espace d'états utilisant cette fois uniquement des DBM [Lime 03a]. Enfin pour les modèles paramétrés, le semi-algorithme de calcul du graphe des classes d'états paramétrées [Traonouez 08] présenté au chapitre 4 est implémenté à l'aide de polyèdres.

Pour les TPN bornés, Romeo implémente une autre méthode de calcul de l'espace d'états : le graphe des zones [Gardey 06], qui converge par inclusion et préserve alors les marquages.

Model-checking en ligne

Grâce au module de calcul Mercutio, Romeo peut vérifier des propriétés temporelles quantitatives (TCTL) sur les modèles de réseaux de Petri temporels. Les formules considérées

appartiennent au sous ensemble TPN-TCTL$_{\mathcal{S}}$ défini au chapitre 4. Bien que restreint ce sous-ensemble est suffisant pour vérifier de nombreuses propriétés intéressantes sur les systèmes temps réel. Des propriétés d'accessibilité peuvent être vérifiées avec des formules du type $\exists\Diamond_{[a,b]}(p)$ (où $[a,b]$ est un interval temporel, avec b éventuellement infini, et p une propriété sur les marquages du réseau) ; des propiétés de sûreté avec $\forall\Box_{[a,b]}(p)$; et également des propriétés de vivacité avec $\forall\Diamond_{[a,b]}(p)$, ou en utilisant l'opérateur de réponse bornée dans une formule de la forme $p \rightsquigarrow_{[0,b]} q$.

L'intérêt de se restreindre à ce sous-ensemble est la possibilité d'utiliser des algorithmes de model-checking « à la volée » basés sur le graphe des classes d'états. Grâce à cette technique il n'est pas nécessaire d'explorer l'ensemble de l'espace d'états pour déterminer la valeur de vérité d'une propriété. Elle est également utilisée sur les automates temporisées dans un outil tel que Uppaal [Larsen 97] et affiche de bonnes performances en pratique. Dans les TPN la vérification est rendue très efficace par l'utilisation de DBM pour encoder les domaines de tir, implémentées à l'aide de la librairie Uppaal DBM Library [Larsen 97]. Dans les réseaux à chronomètres le problème de l'accessibilité est indécidable. Les semi-algorithmes qui sont implémentés utilisent des polyèdres pour encoder les domaines de tirs, avec la bibliothèque Parma Polyhedra Library [Bagnara 08].

Pour les modèles paramétrés Romeo peut vérifier des formules PTCTL$_{\mathcal{S}}$ afin de déterminer les valuations des paramètres pour lesquelles le problème de model-checking est vérifié. Les semi-algorithmes implémentés sont basés sur le graphe des classes d'états paramétrées dans lequel les domaines de tir sont également encodés par des polyèdres. Le résultat fourni par Romeo est un ensemble de contraintes sur les paramètres : une disjonction de polyèdres, également encodée avec la bibliothèque Parma Polyhedra Library à l'aide de « *powerset domain* » [Bagnara 98].

Traduction vers les automates temporisés

Une autre possibilité offerte par Romeo pour vérifier des propriétés sur les réseaux de Petri temporels est de les traduire en automates temporisés, ce qui permet alors d'utiliser les nombreux outils efficaces développés pour ces modèles. Romeo offre pour cela plusieurs traductions des réseaux de Petri temporels en automates temporisés. Tout d'abord par la traduction structurelle proposée dans [Cassez 06b] d'un TPN vers un réseau d'automates temporisés temporellement bisimilaire ; ou bien en utilisant une traduction basée sur un calcul de l'espace d'états, avec soit la méthode de l'automate temporisé des classes d'états [Lime 03b], soit la méthode du graphe des zones [Gardey 06] qui permet de construire un automate temporisé des marquages.

Les réseaux de Petri à chronomètres peuvent quant à eux être traduits en automates à chronomètres [Lime 04b].

5.2 Model-checking paramétré avec Romeo

Les semi-algorithmes de model-checking paramétré présentés au chapitre 4 ont été implémentés dans Romeo, en développant des versions non récursives de ces semi-algorithmes.

Nous proposons également des semi-algorithmes utilisant la méthode de vérification de formules TCTL avec observateur présentée dans [Hadjidj 06]. Au lieu d'étendre le graphe des classes d'états paramétrées du modèle avec une variable supplémentaire, l'intervalle temporel paramétré de la formule PTCTL est transformé en un observateur rajouté au modèle.

5.2.1 Observateur de formules PTCTL

La méthode consiste à intégrer dans le modèle les contraintes temporelles de la formule PTCTL. Dans le cas non paramétré, la vérification d'une formule TCTL peut ainsi être ramenée à la vérification d'une formule non temporisée CTL. Le motif utilisé est présenté sur la figure 5.1 : a et b sont les bornes de l'intervalle temporel paramétré de la formule PTCTL, c'est-à-dire des expressions linéaires sur les paramètres. Ce motif joue le rôle d'un chronomètre permettant de mesurer le temps écoulé. Pour un PTPN $\mathcal{N}$ nous noterons $\mathcal{N}_{obs}$ le modèle obtenu par la composition parallèle de $\mathcal{N}$ avec cet observateur.

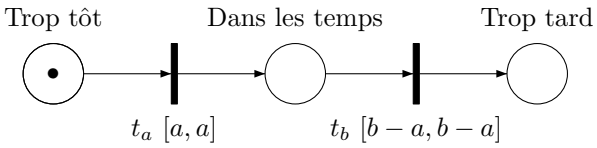


FIG. 5.1: Observateur d'une formule PTCTL

- Dans le cas où l'intervalle temporel paramétré de la formule est de la forme $[a, \infty[$, on utilise un observateur plus simple en supprimant la transition t_b et la place *Trop tard*.
- Dans le cas où $a = 0$, le jeton de la place *Trop tôt* peut être directement déplacé dans la place *Dans les temps*.
- Pour les formules PTCTL de la forme $\varphi \rightsquigarrow_{J_r} \psi$, la place *Trop tôt* n'est jamais marquée puisque $a = 0$, et si $M_0 \not\models \varphi$, la place *Dans les temps* n'est pas non plus marquée initialement. En effet, le chronomètre ne doit pas être démarré tant que φ n'a pas été vérifiée. Il le sera par contre à chaque fois que φ est vérifiée, puis sera réinitialisé après que ψ a été vérifiée. Pour démarrer le chronomètre dans une classe d'états $C = (M, D)$ on ajoute un jeton au marquage M dans la place *Dans les temps* ; pour l'arrêter on enlève dans le marquage M tous les jetons de l'observateur.

Détermination des transitions tirables

L'ajout de ce motif impose de modifier l'algorithme de détermination des transitions tirables dans une classe d'états (et donc le calcul des successeurs) afin qu'il corresponde correctement à la traduction de l'intervalle temporel de la formule. En effet, lorsque la place *Trop tôt* est marquée, le temps écoulé peut tout de même être compris dans l'intervalle temporel si l'horloge de la transition t_a est égale à a (c'est-à-dire si t_a est tirable). De la même manière si la place *Trop tard* est marquée, le temps peut encore être dans l'intervalle temporel à condition que l'horloge de t_b soit égale à 0. En conséquence, on modifie les règles de détermination des transitions tirables en donnant à t_a une *priorité de tir forte* : t_a doit être tirée avant toute transition tirable à la même date ; au contraire, on donne à t_b une *priorité de tir faible* : t_b doit être tirée après toute transition tirable à la même date. Cela est réalisé par l'algorithme 5.1 en ajoutant des contraintes supplémentaires strictes de franchissabilité.

Cet algorithme est utilisé pour calculer les successeurs d'une classe d'états paramétrée. Il détermine les transitions tirables, et les contraintes de franchissabilité ajoutées au domaine de tir sont celles induites par l'algorithme. On notera $\mathsf{succ}_{obs}(C, t)$ le successeur d'une classe d'états paramétrée C, après le tir de t, obtenu par cette méthode. Le graphe des classes d'états paramétrées calculé par cette méthode est alors noté $\mathcal{G}_{obs}$

Algorithme 5.1 : $\mathsf{firable}_{obs}(C, t_f)$

Données : $C = (M, D)$: classe d'états paramétrée, t_f : transition

début
  si $t_f \notin \mathsf{enabled}\,(M)$ **alors retourner** $FAUX$
  si $t_a \in \mathsf{enabled}\,(M)$ ET $t_f \neq t_a$ **alors**
    Soit $D' = D \wedge \left(\bigwedge_{t \in \mathsf{enabled}(M) - \{t_f, t_a\}} (\theta(t_f) \leq \theta(t))\right) \wedge \theta(t_f) < \theta(t_a)$
  sinon si $t_b \in \mathsf{enabled}\,(M)$ ET $t_f = t_b$ **alors**
    Soit $D' = D \wedge \left(\bigwedge_{t \in \mathsf{enabled}(M) - \{t_f\}} (\theta(t_f) < \theta(t))\right)$
  sinon
    Soit $D' = D \wedge \left(\bigwedge_{t \in \mathsf{enabled}(M) - \{t_f\}} (\theta(t_f) \leq \theta(t))\right)$
  fin si
  si $D' \neq \emptyset$ **alors retourner** $VRAI$
  sinon retourner $FAUX$
fin

5.2.2 Semi-algorithmes de model-checking paramétré avec observateurs

Nous proposons des semi-algorithmes de model-checking paramétré avec observateurs équivalents à ceux proposés au chapitre 4. Comme les précédents, chaque semi-algorithme est dédié à un type de formule PTCTL$_{\mathcal{S}}$. Nous les présentons récursivement, de la même manière, en décrivant la formule sur les paramètres qui est calculée pour chaque classe d'états paramétrée. La différence par rapport aux précédents algorithmes est que les conditions temporelles sont ici remplacées par des tests sur les marquages de l'observateur.

5.2.2.1 Semi-algorithme EU-obs

Pour un PITPN $\mathcal{N}$, et une formule PTCTL$_{\mathcal{S}}$ ϕ de la forme $\phi = \exists \varphi \, \mathcal{U}_J \, \psi$, où $J \in \mathcal{J}(Par)$ et $\phi, \psi \in PR$, on détermine le résultat $F^{\phi}(C_0)$ du model-checking paramétré en calculant $F^{\phi}_{\text{EU-obs}}(C_0)$. Le principe du semi-algorithme $F^{\phi}_{\text{EU-obs}}$ est identique à F^{ϕ}_{EU} : les conditions d'accessibilité des classes vérifiant la formule sont ajoutées en disjonction au résultat final.

Soit une classe d'états paramétrée $C = (M, D) \in \mathcal{G}_{obs}(\mathcal{N}_{obs})$, on calcule :

$$F^{\phi}_{\text{EU-obs}}(C) = D_{|Par} \ \wedge \ (M(\text{Trop tard}) = 0) \ \wedge \ \Big(\Big(M \models \psi \ \wedge \ (M(\text{Dans les temps}) = 1) \Big)$$
$$\vee \Big(M \models \varphi \ \wedge \ \mathsf{firable}_{obs}(C) \neq \emptyset \ \wedge \ \Big(\bigvee_{\substack{t \in \mathsf{firable}_{obs}(C) \\ C' = \mathsf{succ}_{obs}(C,t)}} F^{\phi}_{\text{EU-obs}}(C') \Big) \Big) \Big)$$

Nous remarquons qu'il n'est plus nécessaire dans le cas où $M \models \psi$, de distinguer si l'on peut ou non attendre dans la classe C dans le but d'atteindre le temps minimum nécessaire pour vérifier ϕ. Le découpage temporel est en effet automatiquement réalisé par l'observateur.

5.2.2.2 Semi-algorithme AU-obs

Pour un PITPN $\mathcal{N}$, et une formule PTCTL$_\mathcal{S}$ ϕ de la forme $\phi = \forall\varphi\ \mathcal{U}_J\ \psi$, où $J \in \mathcal{J}(Par)$ et $\phi, \psi \in PR$, on détermine le résultat $F^\phi(C_0)$ du model-checking paramétré en calculant $F^\phi_{\text{AU-obs}}(C_0)$. Au contraire de l'algorithme précédent, le principe est d'ajouter en conjonction la négation des conditions d'accessibilité des classes ne vérifiant pas la formule, et ainsi d'empêcher l'accessibilité de ces états.

Soit une classe d'états paramétrée $C = (M, D) \in \mathcal{G}_{obs}(\mathcal{N}_{obs})$, on calcule :

$$F^\phi_{\text{AU-obs}}(C) = D_{|Par} \ \wedge\ (M(\text{Trop tard}) = 0) \ \wedge\ \Big(\Big(M \models \psi \ \wedge\ (M(\text{Dans les temps}) = 1)\Big)$$

$$\vee \Big(M \models \varphi \ \wedge\ \mathsf{firable}_{obs}(C) \neq \emptyset \ \wedge\ \Big(\bigwedge_{\substack{t \in \mathsf{firable}_{obs}(C) \\ C'=(M',D')=\mathsf{succ}_{obs}(C,t)}} (F^\phi_{\text{AU-obs}}(C') \ \vee\ \neg D'_{|Par})\Big)\Big)\Big)$$

5.2.2.3 Semi-algorithme LT-obs

Pour un PITPN $\mathcal{N}$, et une formule PTCTL$_\mathcal{S}$ ϕ de la forme $\phi = \varphi \rightsquigarrow_{J_r} \psi$, où $J_r = [0, b]$ avec $b \in \mathbb{Q}^+ \cup Par$ ou bien $J_r = [0, \infty[$, et $\phi, \psi \in PR$, on considère le modèle $\mathcal{N}_{LT-obs}$, étendu avec la place $M(P_{LT})$ et mis en parallèle avec l'observateur de la formule ϕ. On détermine avec ce modèle le résultat $F^\phi(C_0)$ du model-checking paramétré en calculant $F^\phi_{\text{LT-obs}}(C_0)$.

Soit une classe d'états paramétrée $C = (M, D) \in \mathcal{G}_{obs}(\mathcal{N}_{LT-obs})$, on calcule :

$$F^\phi_{\text{LT-obs}}(C) = D_{|Par} \ \wedge\ \Big(M(P_{LT}) = 0 \ \vee\ (M(\text{Dans les temps}) = 1)\Big)$$

$$\wedge \Big(\Big(\mathsf{firable}_{obs}(C) = \emptyset \ \wedge\ (M(P_{LT}) = 0 \ \vee\ M \models \psi)\Big) \vee$$

$$\Big(\mathsf{firable}_{obs}(C) \neq \emptyset \ \wedge\ \Big(\bigwedge_{\substack{t \in \mathsf{firable}_{obs}(C) \\ C'=(M',D')=\mathsf{succ}_{LT-obs}(C,t)}} (F^\phi_{\text{LT-obs}}(C') \ \vee\ \neg D'_{|Par})\Big)\Big)\Big)$$

Le calcul des successeurs succ_{LT-obs} intègre la modification des contraintes temporelles afin de gérer l'observateur, ainsi que la mise à jour de la place P_{LT} du modèle $\mathcal{N}_{LT}$ et le redémarrage du chronomètre, tels qu'ils sont décrits au chapitre 4.

5.2.2.4 Convergence par inclusion

Nous pouvons montrer, de la même manière que pour les algorithmes du chapitre 4, que si une classe d'états paramétrée C'' est incluse dans une classe C' alors :

- $F^\phi_{\text{EU-obs}}(C'') \subseteq F^\phi_{\text{EU-obs}}(C')$,
- $(F^\phi_{\text{AU-obs}}(C') \vee \neg D'_{|Par}) \subseteq (F^\phi_{\text{AU}}(C'') \vee \neg D''_{|Par})$,
- $(F^\phi_{\text{LT-obs}}(C') \vee \neg D'_{|Par}) \subseteq (F^\phi_{\text{LT}}(C'') \vee \neg D''_{|Par})$.

Nous pouvons donc à nouveau appliquer un critère de convergence par inclusion de classes d'états paramétrées. Dans le cas où une classe C'' est incluse dans une classe C', mais telle que C'' n'est pas un successeur de C', les ajouts des conditions pour C'', en disjonction pour l'algorithme EU, et en conjonction pour les algorithmes AU et LT, sont inutiles car ces conditions sont alors redondantes par rapport aux conditions déjà calculées pour C'.

L'utilisation du critère d'inclusion peut être généralisée, pour éviter d'examiner une classe C'' incluse dans une autre classe C', même lorsque la classe C'' est un successeur de C'. Des précautions doivent cependant être prises pour déterminer les actions à effectuer.

1. Dans l'algorithme EU, il suffit de ne pas analyser $F^{\phi}_{\mathrm{EU}}(C'')$: en effet pour toute exécution correcte comportant C' et C'', il existe une autre exécution partant de C', mais ne comportant pas C'', qui est correcte et avec une condition d'accessibilité moins stricte.
2. Dans les algorithmes AU et LT, dans le cas où il y a égalité entre les deux classes, on forme alors un cycle dans le graphe des classes d'états qui peut être emprunté une infinité de fois. Cela correspond en particulier aux cas zénons. Les exécutions correspondant à ce cycle sont incorrectes, il faut donc empêcher l'accessibilité de la classe C'. Dans le cas général, il faut s'assurer que si $C'' \subset C'$, il existe bien un autre chemin partant de C' aboutissant à un état vérifiant ψ. Dans le cas contraire il faut à nouveau empêcher l'accessibilité de la classe C'.

5.2.3 Réduction des domaines des tirs

Au cours du model-checking paramétré, nous calculons les valeurs des paramètres admissibles pour vérifier la formule. Le résultat intermédiaire, duquel sera issue le résultat final, peut être utilisé pour modifier en direct, en cours d'analyse, le domaine des paramètres du PITPN étudié. On peut ainsi réduire les domaines de tir des classes restant à explorer, afin de ne considérer que les valeurs des paramètres nécessaires.

1. Dans l'algorithme EU, $F^{\phi}(C_0)$ est initialement vide et les valeurs des paramètres vérifiant le problème sont ajoutées en disjonction au fur et à mesure de l'analyse. On peut alors restreindre les domaines de tir des classes à explorer en ne considérant que les valeurs de paramètres qui ne sont pas encore dans le résultat, c'est-à-dire dans $\neg F^{\phi}(C_0)$.
2. Dans les algorithmes AU et LT au contraire, le résultat $F^{\phi}(C_0)$ est initialisé à D_p, et il est restreint au fur et à mesure de l'analyse en éliminant les valeurs incorrectes. Cela nous permet de restreindre les domaines de tir des classes à explorer en éliminant ces valeurs incorrectes des paramètres, et donc en ne gardant que celles qui sont dans $F^{\phi}(C_0)$.

Puisque $F^{\phi}(C_0)$ (et respectivement $\neg F^{\phi}(C_0)$) est une disjonction de polyèdres, pour pouvoir contraindre les polyèdres des domaines de tir des classes restant à explorer, il est par contre nécessaire de séparer les polyèdres en disjonction dans $F^{\phi}(C_0)$ (resp. $\neg F^{\phi}(C_0)$). On est donc amené à dupliquer les classes suivantes selon le polyèdre de contraintes choisi dans $F^{\phi}(C_0)$ (resp. $\neg F^{\phi}(C_0)$).

L'utilisation de cette technique dans Romeo a amélioré l'efficacité du model-checking paramétré qui est limitée par la taille du domaine des paramètres. On diminue grâce à elle la dépendance entre la complexité du model-checking paramétré et la taille du domaine initial des paramètres. Il est toutefois toujours intéressant de restreindre le domaine des paramètres par des contraintes supplémentaires, lorsqu'elles peuvent empêcher l'accessibilité de certaines parties du graphe des classes d'états paramétrées.

5.2.4 Obtention de conditions suffisantes

Les semi-algorithmes de model-checking paramétré proposés résolvent le problème de synthèse des paramètres dans lequel on cherche à déterminer l'ensemble des valeurs correctes des

paramètres, c'est-à-dire une condition nécessaire et suffisante sur les paramètres, mais ce problème est indécidable. Dans les cas où aucun résultat à ce problème ne peut être déterminé, des résultats partiels peuvent toutefois être obtenus, sous la forme de conditions nécessaires ou suffisantes. Ainsi :

1. Dans le semi-algorithme EU, si l'on arrête l'analyse avant le terme, l'ensemble de contraintes $F^{\phi}(C_0)$ contient des conditions suffisantes sur les paramètres pour vérifier la formule ϕ.
2. Dans les semi-algorithmes AU et LT au contraire, avant la fin de l'analyse, $F^{\phi}(C_0)$ contient des conditions nécessaires sur les paramètres.

En se servant des ces propriétés, Romeo est capable de déterminer des conditions suffisantes pour vérifier une propriété PTCTL. L'algorithme utilise pour cela une condition d'arrêt sur la taille des traces à explorer. Les semi-algorithmes sont légèrement modifiés afin de retourner des conditions suffisantes quelle que soit la forme de la formule PTCTL :

1. L'algorithme EU retourne normalement une condition suffisante pour les formules $\exists \varphi \, \mathcal{U}_J \, \psi$ et l'abréviation $\exists \Diamond_J \varphi$. Il est également utilisé en calculant la négation pour vérifier les formules $\forall \Box_J \varphi$, mais retourne dans ce cas une condition nécessaire. Pour retourner une condition suffisante, au lieu de simplement stopper l'exploration d'une trace trop longue, nous ajoutons au résultat la condition d'accessibilité de la classe finale, comme si la formule était vérifiée.
2. Inversement, l'algorithme AU retourne quant à lui une condition nécessaire. Après calcul de la négation, elle se transforme en condition suffisante pour les formules $\exists \Box_J \varphi$. Pour les formules $\forall \varphi \, \mathcal{U}_J \, \psi$ et $\forall \Diamond_J \varphi$, il est nécessaire lors de l'arrêt de l'exploration d'une trace trop longue de rajouter au résultat la négation de la condition d'accessibilité de la dernière classe, comme cela est normalement réalisé pour les classes ne vérifiant pas la formule. Cela est également le cas pour l'algorithme LT.

Romeo peut donc dans tous les cas retourner des conditions suffisantes sur les paramètres afin de vérifier une formule PTCTL. On note qu'en utilisant la négation d'une formule on pourra également obtenir des conditions nécessaires.

5.3 Étude de cas

Les fonctionnalités de Romeo sur les modèles paramétrés présentées dans ce chapitre, sont maintenant illustrées sur un exemple d'ordonnancement de tâches issue de [Bucci 04a] (repris dans [Berthomieu 05]). Nous considérons un système de trois tâches périodiques : la tâche 1 a une période de a unités de temps, la tâche 2 est sporadique avec un délai d'inter-arrivée de $2a$ unités de temps, et la tâche 3 est périodique de période $3a$. Le système est donc paramétré par un paramètre temporel a commun aux trois tâches. Les trois tâches sont dites à *échéances sur requêtes*, c'est-à-dire que les échéances auxquelles les tâches doivent être terminées coïncident avec leur activation suivante. Par ailleurs un ordonnancement à priorités fixes existe entre les trois tâches : la tâche 1 a la plus forte priorité, ensuite la tâche 2 et enfin la tâche 3 a la priorité la plus faible.

Ce système est modélisé dans Romeo par un réseau de Petri temporel avec arcs inhibiteurs, présenté dans la fenêtre d'édition de Romeo sur la figure 5.2. Le modèle des tâches comporte pour chacune d'elles trois transitions :

- T_{i1} est l'activation de la tâche i, associée à un intervalle temporel paramétré égal à la période. Sa place de sortie P_{i1} correspond donc à l'état *active*.
- T_{i2} est la terminaison de la tâche après une durée d'exécution variable qui est fixée sur l'intervalle temporel de la transition. Elle produit un jeton dans la place P_{i2} qui correspond donc à l'état *terminée*.
- T_{i3} est utilisée pour vider immédiatement le jeton produit en P_{i2} afin que le modèle soit borné, son intervalle temporel est donc $[0, 0]$.

Les arcs inhibiteurs sont utilisés pour modéliser les priorités entre les tâches :

- Lorsque la tâche 1 est *active* elle doit préempter l'exécution des deux autres tâches ; un arc inhibiteur relie donc la place P_{11} aux transitions T_{22} et T_{32} afin d'inhiber l'exécution des deux autres tâches.
- La tâche 2 inhibe quant à elle l'exécution de la tâche 3 avec un arc inhibiteur de P_{21} vers T_{32}.

On peut également rajouter des contraintes sur le paramètre a : la fenêtre de rajout des contraintes apparaît sur la figure 5.2 ; on restreint ici le domaine du paramètre de sorte que : $D_p = \{30 \leq a \leq 70\}$. On aurait pu également modéliser ce système avec ROMEO en traduisant directement les priorités à l'aide de l'extension à l'ordonnancement (Scheduling-PTPN). puisque nous n'avons présenté que les extensions avec arcs inhibiteurs nous conservons cette modélisation.

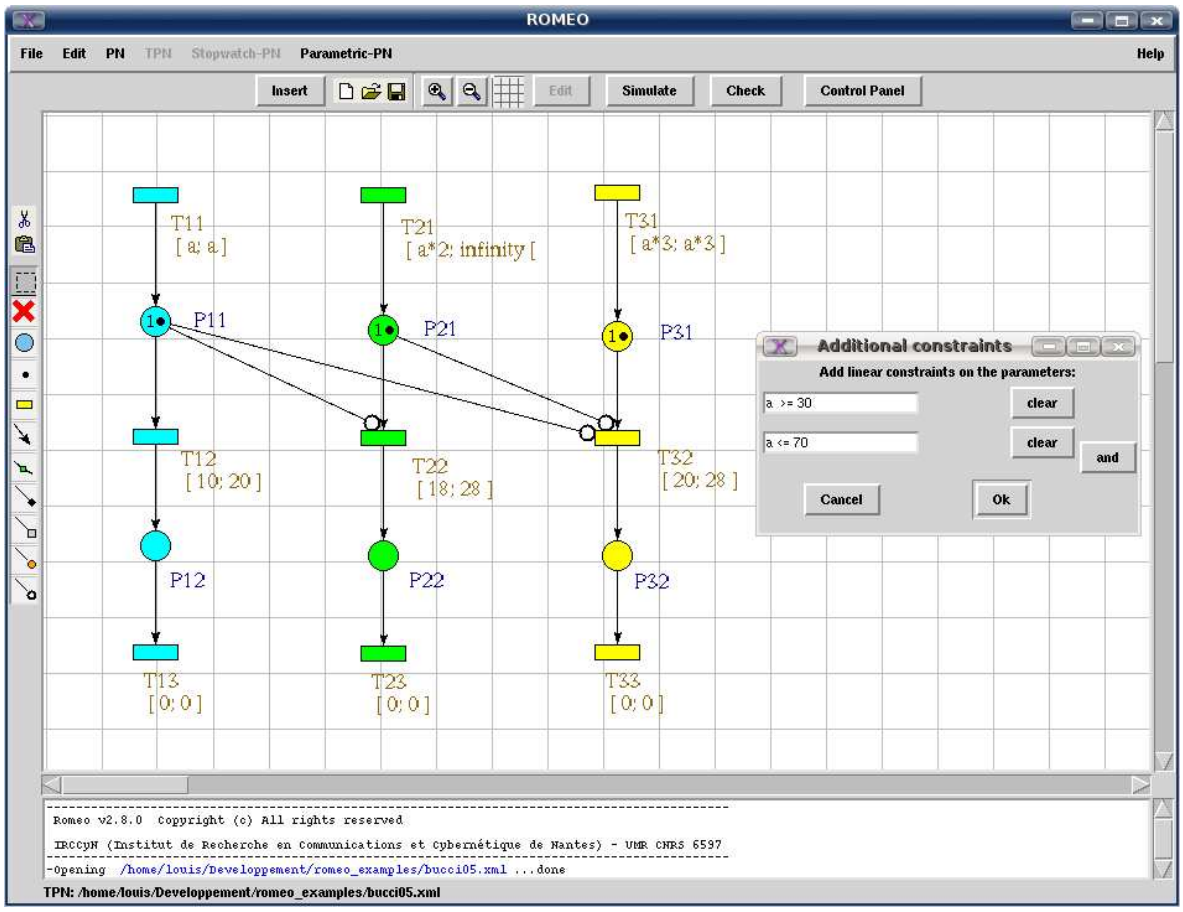


FIG. 5.2: Interface graphique de ROMEO avec un modèle PITPN de trois tâches périodiques

Vérification

Une fois le système modélisé dans ROMEO nous pouvons entreprendre la vérification de propriétés sur le modèle. Parmi les problèmes intéressants sur ce système, la première question concerne l'*ordonnançabilité* des trois tâches : « *pour chaque instance d'une tâche, le temps de réponse de la tâche (i.e. le délai entre l'activation et la terminaison de la tâche) doit être inférieur à son échéance (i.e. son activation suivante dans cet exemple à échéances sur requêtes)* ». On vérifie cette propriété en vérifiant si le modèle est sauf (*i.e.* 1-borné) : en effet les tâches ne seront pas ordonnançables si l'une d'entre elles (une tâche i) peut produire un jeton dans la place P_{i1} déjà marquée (*i.e.* son activation a lieu avant la terminaison de l'instance précédente). Cette propriété peut être vérifiée dans ROMEO avec la formule PTCTL suivante, pour $i \in \{1,2,3\}$ et $j \in \{1,2\}$:

$$\forall P_{ij},\ \forall\Box_{[0,\infty[}(M(P_{ij}) \leq 1)$$

Le résultat obtenu par model-checking paramétré avec ROMEO fournit la contrainte suivante sur le paramètre a :

$$\{a > 48\}$$

Ce résultat nous permet alors de rajouter la contrainte obtenue dans les contraintes supplémentaires ajoutées aux paramètres, de sorte que maintenant : $D_p = \{48 < a \leq 70\}$. En faisant cela on s'assure que le modèle paramétré est maintenant ordonnançable, pour toutes les valeurs de a comprises dans D_p.

Puisque le système est ordonnançable, on peut vérifier de nouvelles propriétés, et notamment calculer le pire temps de réponse des tâches (WCRT, pour *Worst Case Response Time*). Pour une tâche i son WCRT est l'instance de cette tâche ayant le temps de réponse le plus important. On vérifie pour cela la propriété PTCTL suivante :

$$(M(P_{31}) > 0) \rightsquigarrow_{[0,b]} (M(P_{32}) > 0)$$

Cette propriété paramétrée permet de calculer le WCRT de la tâche 3, à l'aide d'un nouveau paramètre b. En effet, cette formule vérifie que le temps de réponse entre les états « P_{31} marquée » et « P_{32} » est toujours inférieur à b. Le pire cas sera nécessairement lorsque le jeton en P_{31} vient tout juste d'arriver, donc lorsque la tâche 3 vient d'être activée. Quant à P_{32} l'arrivée du jeton coïncide avec son départ à cause de la transition T_{33} qui doit être tirée immédiatement, ce qui correspond à la fin de la tâche 3. Ainsi les valeurs acceptables pour b sont nécessairement supérieures au pire temps de réponse, donc b est une borne supérieure du WCRT. La valeur la plus faible sera alors égale au WCRT. Or le résultat obtenu par ROMEO pour la vérification de cette formule est :

$$\{b \geq 96\}$$

On en déduit donc que le pire temps de réponse de la tâche 3 est 96. Ce résultat bien qu'indépendant du paramètre a est valable pour toutes les valeurs de a comprises entre 48 et 70. Il est conforme au résultat obtenu dans l'article [Berthomieu 05], dans lequel pour la valeur $a = 50$ les auteurs calculent le même WCRT. Dans cet article le calcul du WCRT est réalisé par un observateur construit pour la valeur 96. L'intérêt de l'approche paramétrée est ici de pouvoir calculer une valeur sans connaissance *a priori*, ce qui peut s'utiliser même sur des modèles non paramétrés.

5.4 Conclusion

Nous avons présenté dans ce chapitre l'utilisation du logiciel Romeo avec les réseaux de Petri temporels paramétrés. La plupart des fonctionnalités du logiciel a été étendue aux modèles paramétrés.

Cela inclut notamment le model-checking paramétré pour lequel deux implémentations ont été testées : la première implémente les semi-algorithmes présentés au chapitre 4, c'est-à-dire en ajoutant une variable supplémentaire aux domaines de tir des classes d'états paramétrées ; la deuxième utilise un observateur pour traduire la formule TCTL paramétrée. Cette dernière méthode s'est révélée plus efficace à l'implémentation car elle facilite l'application de méthodes d'optimisation. La convergence par inclusion y est notamment facilitée : dans la première méthode, l'ajout de la variable supplémentaire aux domaines de tir pour compter le temps écoulé empêche en effet dans de nombreuses situations l'inclusion des domaines de tir entre les classes.

Bien que les problèmes intéressants soient indécidables sur les modèles paramétrés, plusieurs problèmes décidables présentées dans ce chapitre permettent d'extraire des conclusions sur un modèle paramétré. Tout d'abord le simulateur de Romeo permet d'explorer à la main une trace dans le graphe des classes d'états paramétrées du modèle. Il fournit ainsi les conditions d'accessibilité sur les paramètres des classes atteintes. Il est ensuite possible d'extraire et d'utiliser les résultats intermédiaires du model-checking paramétré .

Nous avons enfin présenté une étude de cas illustrant l'intérêt d'une approche paramétrée. Un premier paramètre est introduit dans le modèle : sa valeur est contrainte par model-checking paramétré afin de ne garder que les valuations des paramètres vérifiant une spécification. Le model-checking paramétré permet donc d'interagir avec le modèle et se rapproche ainsi plus de la synthèse que de la vérification. Dans un deuxième temps, un second paramètre est introduit dans la formule TCTL à vérifier. Ce cas d'utilisation peut s'utiliser même sur modèles non paramétrés pour lesquels on veut déterminer des spécification temporelles (dans cet exemple un temps de réponse).

Troisième partie

Dépliage paramétré

Chapitre 6

Méthode de dépliage de réseaux de Petri temporels paramétrés

Résumé : *Le parallélisme naturel des réseaux de Petri n'est pas conservé par les méthodes traditionnelles d'analyse telles que le model-checking. C'est une des raisons qui peut amener au problème de l'explosion combinatoire.*
Les dépliages sont des méthodes d'analyse alternatives qui proposent de conserver un parallélisme « vrai ». Nous proposons une nouvelle méthode de dépliage temporel de réseaux de Petri temporels (éventuellement paramétrés). Elle se base sur une analyse des situations de conflits et présente l'intérêt de fournir une structure dépliée la plus compacte possible. Avant de décrire la méthode proposée, nous présenterons un récapitulatif des travaux sur les dépliages de réseaux de Petri (temporels et non temporels) et nous introduirons les notions classiques utilisées dans les dépliages.

Sommaire

6.1 Méthodes de préservation de la concurrence

Le parallélisme, c'est-à-dire la concurrence des évènements, s'exprime naturellement dans les réseaux de Petri. Le fait de pouvoir utiliser plusieurs jetons permet par exemple de représenter les états de plusieurs processus s'exécutant en parallèle. Un état dans un réseau de Petri, c'est-à-dire un marquage, correspond alors à la composition des états de chacun des processus. C'est une propriété spécifique aux réseaux de Petri qui se distinguent notamment par ce biais des automates dans lesquels un état est limité à une seule localité.

Dans les réseaux de Petri temporels, malheureusement, les méthodes de construction de l'espace d'états (graphe des classes d'états, graphe des zones...) ne conservent pas ce que l'on nomme le parallélisme « vrai ». Ces méthodes se basent en effet sur une sémantique séquentielle des réseaux de Petri temporels, exprimée sous la forme d'un système de transitions temporisé, dans laquelle les évènements concurrents sont entrelacés. Ainsi, deux évènements concurrents e_1, e_2 d'un réseau de Petri (des tirs de transitions) sont entrelacés dans les deux séquences de transitions $s_1 = e_1; e_2$, lorsque le tir de e_1 arrive avant e_2, et $s_2 = e_1; e_2$, si au contraire e_2 arrive avant e_1.

La préservation de la concurrence dans les réseaux de Petri nécessite donc de nouvelles méthodes d'analyse. Les buts recherchés sont multiples : l'amélioration des performances en limitant l'explosion combinatoire due aux entrelacements, la possibilité d'effectuer des analyses sur des parties du réseau ou encore l'analyse des ordres partiels entre les évènements. On distingue deux approches différentes proposées afin d'atteindre un ou plusieurs de ces objectifs. Dans les deux cas le principe de base est de conserver les ordres partiels existant entre les évènements, plutôt qu'entrelacer en instaurant un ordre total.

Une première catégorie de méthodes, appelées réductions des ordres partiels, consiste à détecter les entrelacements inutiles lors de l'exploration de l'espace d'états afin de limiter l'explosion combinatoire. Diverses méthodes peuvent être utilisées selon le type de propriétés que l'on souhaite préserver ; on pourra trouver un panorama complet dans [Godefroid 96]. Dans les réseaux de Petri temporels on peut citer en particulier la méthode du graphe des pas couvrant [Vernadat 96] préservant les propriétés linéaires de type LTL, et celle proposée dans [Penczek 01] à l'aide du graphe des classes d'états géométriques qui préserve les propriétés de branchement CTL*.

L'autre approche consiste à définir une sémantique concurrente pour les réseaux de Petri, c'est-à-dire une sémantique préservant les ordres partiels. Dans les réseaux de Petri classiques, elle est définie en terme de *processus* [Best 88] ou de *processus de branchement* [Nielsen 80, Engelfriet 91]. Un processus décrit une exécution concurrente dans un réseau de Petri, ce qui correspond à un ensemble de séquences de transitions, équivalentes aux entrelacements près. Il définit donc un ordre partiel entre les évènements, que l'on interprète comme des relations de causalité entre les évènements. Les processus de branchement regroupent alors un ensemble de processus liés par une relation de conflit. Ils permettent de définir le *dépliage* d'un réseau de Petri, qui est le plus grand processus de branchement constructible (en général infini), également défini comme l'ensemble des processus du réseau de Petri. Un algorithme de model-checking utilisant cette méthode de préservation de la concurrence est proposé dans [McMillan 95], et amélioré dans [Esparza 96]. Il consiste à construire un préfixe fini complet du dépliage du réseau de Petri qui contient l'ensemble des marquages accessibles. La méthode peut également être utilisée pour vérifier des propriétés de logique temporelle [Esparza 94].

Cette technique de dépliage a depuis lors connu de nombreux développements (un exposé complet de la théorie des dépliages est présenté dans [Esparza 08]). Elle a notamment été appliquée à d'autres classes plus expressives de réseaux de Petri telles que les réseaux non bornés [Abdulla 00] ou des réseaux de Petri de haut niveau [Khomenko 03].

Dans les réseaux de Petri temporels, une sémantique concurrente est définie dans [Aura 97] à l'aide de processus temporels. Un processus temporel associe à un processus une fonction de temporisation qui donne les dates de tir des évènements. Les auteurs présentent une méthode pour déterminer l'ensemble des temporisations valides pour un processus. La définition du dépliage d'un TPN, c'est-à-dire le calcul de l'ensemble des processus temporels est quant à elle donnée par les auteurs de [Chatain 06b]. Ils doivent pour cela étendre la définition des processus temporels pour intégrer des causalités supplémentaires induites par l'introduction du temps. Ils définissent également un préfixe fini complet au dépliage, suffisant pour construire l'ensemble des processus temporels du TPN. Cette méthode de dépliage a également été appliquée aux réseaux d'automates temporisés [Cassez 06a].

Enfin, une approche similaire aux dépliages est basée sur la modélisation des réseaux de Petri en formules de logique linéaire. La logique linéaire, introduite par [Girard 87], est une logique non monotone exprimant le changement d'état, c'est-à-dire la disparition et/ou la création de ressources, ainsi que l'aspect quantitatif des ressources. Les travaux décrits dans [Girault 97] ont montré qu'un sous-ensemble des formules de cette logique, appelé fragment MILL, permet de représenter les réseaux de Petri. Par un processus de preuves de formules de logique linéaire dans le calcul des séquents, il est en effet possible de vérifier l'accessibilité d'un marquage dans un réseau de Petri. Ce processus étant indépendant de l'ordonnancement des transitions concurrentes il permet de construire un processus. L'équivalence entre ces preuves de formules linéaires et les processus des réseaux de Petri est étudiée en détail dans [Rivière 03].

Cette méthode a été étendue pour vérifier des propriétés temporelles quantitatives dans les TPN. La technique consiste à annoter la preuve d'un séquent correspondant à un scénario de tir dans le réseau de Petri afin de déterminer des contraintes sur les dates de tir des évènements. D'abord limitée à la sémantique faible des TPN [Pradin-Chézalviel 99], nous avons étendu la méthode à la sémantique forte [Delfieu 07].

Nous présentons plus en détails dans la suite de cette partie certains des travaux précédemment cités sur les dépliages et que nous réutilisons dans notre méthode de dépliage. Nous définissons notamment les différentes notions utilisées, en se basant sur les définitions des dépliages de réseaux de Petri données par [Esparza 96].

Pour illustrer les propos, nous nous servirons du réseau de Petri T-temporel de la figure 6.1, issue de [Chatain 06b]. Dans un premier temps, on ne considérera que le réseau de Petri non temporel sous-jacent en ignorant les temporisations des transitions.

6.1.1 Réseaux d'occurrences

Les processus et les processus de branchement dans les réseaux de Petri peuvent être représentés par une structure possédant la même syntaxe place/transition que les réseaux de Petri eux-mêmes.

Nous rappelons qu'un réseau est un triplet $\langle S, T, W \rangle$ définissant un graphe biparti orienté, où les deux types de nœuds sont les places S et les transitions T, reliés entre eux par des arcs

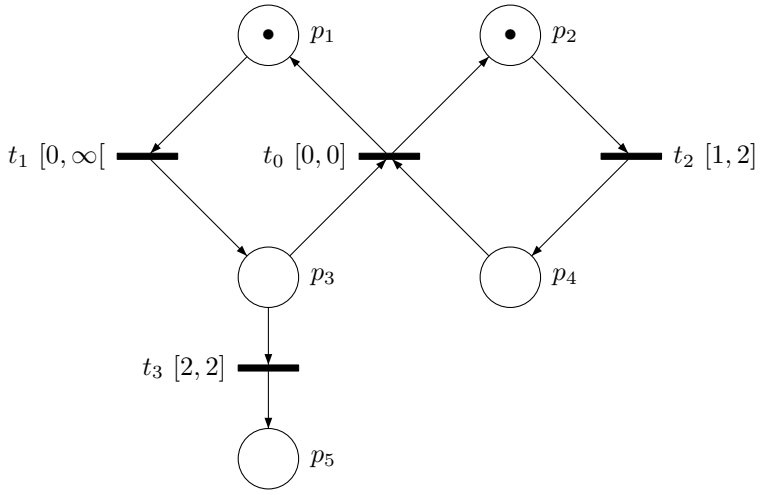


FIG. 6.1: Un réseau de Petri temporel avec une boucle

définis par la relation W. W est une fonction $(S \times T) \cup (T \times S) \to \{0,1\}$; pour deux nœuds x et y, un arc relie x à y si $W(x,y) = 1$. On se limite donc ici à des arcs de poids égaux à 1. Comme pour les réseaux de Petri, on peut définir les fonctions d'incidence amont et aval d'un nœud x, par ${}^\bullet x = \{y \in S \cup T \mid W(y,x) = 1\}$, et $x^\bullet = \{y \in S \cup T \mid W(x,y) = 1\}$. Un chemin dans le graphe sous-jacent à un réseau est une séquence $x_1 x_2 \dots x_n$ tel que $\forall i \in [1..n-1],\ W(x_i, x_{i+1}) = 1$.

On définit dans un réseau différentes relations de causalité entre deux nœuds $x, y \in S \cup T$:

- x et y sont en *relation causale*, ce que l'on note $x < y$, ssi il existe un chemin, constitué par au moins un arc, de x vers y. On dira que x *précède* y.
- x et y sont en *conflit*, ce que l'on note $x \# y$, ssi il existe deux chemins $st \dots x$ et $su \dots y$, partant de la même place $s \in S$, mais tels que $t \neq u$.
- x et y sont en *concurrence*, ce que l'on note x co y, ssi aucune des deux relations précédentes n'est vérifiée, c'est-à-dire ni $x < y$, ni $y < x$, ni $x \# y$.

Les réseaux d'occurrences sont un type particulier de réseaux qui sert à définir les processus de branchement, et donc le dépliage d'un réseau de Petri. Ils définissent un ordre partiel entre les nœuds du réseau.

Définition 6.1 (Réseau d'occurrences) *Un* réseau d'occurrences *est un réseau* $\mathcal{O} = \langle B, E, F \rangle$ *tel que :*

- *pour tout* $b \in B,\ |{}^\bullet b| \leq 1$,
- $\mathcal{O}$ *est acyclique, c'est-à-dire que la relation causale est un ordre partiel,*
- $\mathcal{O}$ *est fini par précédence, c'est-à-dire que pour tout* $x \in B \cup E$, *l'ensemble des éléments* $y \in B \cup E$ *tels que* $y < x$ *est fini,*
- *aucun élément n'est en conflit avec lui-même.*

Dans un réseau d'occurrences, les places B de $\mathcal{O}$ sont appelées des *conditions*, et les transitions de E sont des *évènements*. S'il existe, l'évènement ${}^\bullet b$ d'une condition b pourra être appelé *producteur* de b, et les conditions de $e^\bullet$ sont les *conditions produites* par l'évènement e. De la même manière, pour une condition b, les évènements de $b^\bullet$ sont les *consommateurs*

de b, et pour un évènement e, les conditions de ${}^\bullet e$ sont les *conditions consommées* par e.

6.1.2 Processus de branchement et dépliage d'un réseau de Petri

Les processus de branchement sont des réseaux d'occurrences utilisés pour déplier un réseau de Petri. Pour cela, une fonction d'étiquetage associe à chaque condition une place du réseau de Petri initial, et à chaque évènement une transition.

Définition 6.2 (Processus de branchement) *Un* processus de branchement *d'un réseau de Petri* $\mathcal{N} = \langle P, T, {}^\bullet(), ()^\bullet, M_0\rangle$ *est un réseau d'occurrences étiqueté* $\beta = \langle \mathcal{O}, l\rangle$, *où* $\mathcal{O} = \langle B, E, F\rangle$ *est un réseau d'occurrences, et* l *est la* fonction d'étiquetage *telle que :*

- $l(B) \subseteq P$ *et* $l(E) \subseteq T$,
- *pour tout* $e \in E$, *la restriction de* l *à* ${}^\bullet e$ *est une bijection entre* ${}^\bullet e$ *et* ${}^\bullet l(e)$, *ainsi qu'entre* $e^\bullet$ *et* $l(e)^\bullet$,
- *pour tout* $e_1, e_2 \in E$ *si* ${}^\bullet e_1 = {}^\bullet e_2$ *et* $l(e_1) = l(e_2)$ *alors* $e_1 = e_2$.

On définit un évènement spécial $\perp \in E$ *qui initialise le dépliage avec le marquage initial du réseau de Petri, de sorte que*

- ${}^\bullet \perp = \emptyset$,
- $l(\perp) = \emptyset$,
- *la restriction de* l *à* $\perp^\bullet$ *est une bijection entre* $\perp^\bullet$ *et* M_0,

Deux exemples de processus de branchement issus du réseau de Petri de la figure 6.1 sont présentés sur la figure 6.2.

Les processus de branchement se distinguent par le niveau du dépliage qu'ils proposent. Ainsi, le processus de branchement 6.2a déplie une fois la boucle t_1, t_2, t_0 du réseau. Le dépliage complet du réseau est infini puisqu'il serait nécessaire de déplier entièrement la boucle.

Préfixe On peut comparer les processus de branchement entre eux selon une relation d'ordre partiel définissant un préfixe. Soient $\beta = \langle \mathcal{O}, l\rangle$ et $\beta' = \langle \mathcal{O}', l'\rangle$ deux processus de branchement, β' est un préfixe de β si et seulement si :

- $\mathcal{O}'$ est un sous-réseau de $\mathcal{O}$,
- pour toute condition $b \in \mathcal{O}'$, son producteur ${}^\bullet b \in \mathcal{O}$ appartient aussi à $\mathcal{O}'$,
- pour tout évènement $e \in \mathcal{O}'$, les conditions consommées et produites par e (${}^\bullet e \cup e^\bullet \subset \mathcal{O}$) appartiennent aussi à $\mathcal{O}'$,
- l' est la restriction de l à $\mathcal{O}'$.

Par exemple, le processus de la figure 6.2b est un préfixe du processus de branchement 6.2a.

Définition 6.3 (Dépliage d'un réseau de Petri) *Il est montré dans [Engelfriet 91] qu'un réseau de Petri* $\mathcal{N}$ *possède un seul et unique processus de branchement maximal selon la relation de préfixe. Ce processus de branchement maximal constitue le* dépliage *du réseau de Petri. On notera* $\mathcal{U}(\mathcal{N})$ *le dépliage de* $\mathcal{N}$.

6.1.2.1 Configurations et coupures

Les processus de branchement ont été introduits afin de représenter l'ensemble des processus d'un réseau de Petri, un processus étant une exécution concurrente dans le réseau de Petri. Ces processus, que l'on nommera également *configurations*, sont extraits d'un processus de branchement par la définition suivante :

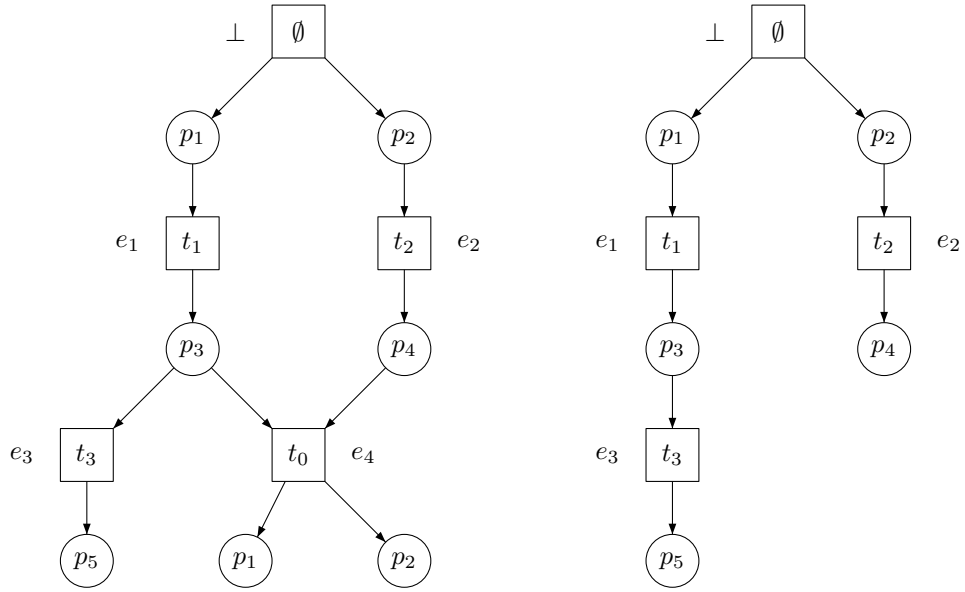


FIG. 6.2: Deux processus de branchement du réseau de Petri de la figure 6.1

Définition 6.4 (Configuration) *Une* configuration *(ou* processus*) d'un processus de branchement $\beta = \langle B, E, F, l\rangle$ est un ensemble E' d'évènements de β tel que :*

- *pour tout $e' \in E'$, $\forall e \in E,\ e < e' \Rightarrow e \in E'$ (E' est causalement clos),*
- *pour tous $e, e' \in E'$ $\neg(e \# e')$ (E' est sans conflit).*

On notera $E'^{\bullet}$ l'ensemble des conditions produites par les évènements d'une configuration E', et ${}^{\bullet}E'$ l'ensemble des conditions consommées. Puisque E' est causalement clos, on remarque que nécessairement ${}^{\bullet}E' \subseteq E'^{\bullet}$.

On remarque aussi qu'une configuration définit un processus de branchement lorsque l'on ajoute les conditions produites par l'ensemble de ses évènements (c'est-à-dire que pour une configuration E' de $\beta = \langle B, E, F, l\rangle$, $\beta_{E'} = \langle E'^{\bullet}, E', F_{|E'}, l_{|E'}\rangle$ est un processus de branchement, où $F_{|E'}$ et $l_{|E'}$ sont les restrictions de F et l à $E' \cup E'^{\bullet}$; de plus $\beta_{E'}$ constitue un préfixe de β). On peut donc représenter une configuration de la même manière par le graphe d'un réseau.

Ainsi, le processus de branchement de la figure 6.2b définit également la configuration $\{\bot, e_1, e_2, e_3\}$ du processus de branchement 6.2a.

Co-set et coupures Les notions de co-set et de coupure vont permettre de retrouver les marquages accessibles du réseau de Petri.

- Un co-set dans un processus de branchement est un ensemble de conditions $B' \subseteq B$ en relation par co par paire (*i.e.* en concurrence). Les conditions ${}^{\bullet}e$ consommées par un évènement e constituent notamment un co-set.
- Une *coupure* est un co-set maximal au sens de l'inclusion.

À un co-set ou une coupure B' on associe un marquage M, tel que pour toute place $p \in P$ du réseau de Petri, $M(p)$ est égal au nombre de conditions dans B' étiquetées par la place p. On notera $l(B')$ ce marquage. Pour un co-set non maximal ce marquage n'est que partiel.

Il est prouvé dans [Best 88, Engelfriet 91] que les coupures du dépliage $\mathcal{U}(\mathcal{N})$ correspondent exactement aux marquages accessibles du réseau de Petri $\mathcal{N}$.

Pour une configuration E' d'un processus de branchement, le co-set $\mathsf{Cut}\,(E')$ défini ci-dessous est une coupure représentant le marquage final obtenu après l'exécution du processus.

$$\mathsf{Cut}\,(E') = E'^{\bullet} \setminus {}^{\bullet}E'$$

Configuration locale Finalement, pour un évènement e d'un processus de branchement β, une configuration particulière est la *configuration locale* de e, notée $[e]$, définie par :

$$[e] = \{e' \in \beta \mid e' \leq e\}$$

$[e]$ contient l'ensemble du passé causal de e.

6.1.2.2 Extensions

Les *extensions* d'une configuration E' correspondent aux transitions sensibilisées par le marquage final atteint en $\mathsf{Cut}\,(E')$. Une extension e est une paire $\langle t, {}^{\bullet}e\rangle$, consituée d'une transition t (par extension on pourra noter $t = l(e)$) et d'un ensemble de préconditions ${}^{\bullet}e$. On note $PE(E')$ l'ensemble des extensions de E' ; $e \in PE(E')$ ssi

- $t \in \mathsf{enabled}\,(l(\mathsf{Cut}\,(E')))$ (t est sensibilisée par le marquage final de E'),
- ${}^{\bullet}e \subseteq \mathsf{Cut}\,(E')$ (e consomme uniquement des conditions finales),
- la restriction de l à ${}^{\bullet}e$ est une bijection entre ${}^{\bullet}e$ et ${}^{\bullet}t$ (les conditions ${}^{\bullet}e$ consommées sont exactement celles qui sensibilisent t).

Nous définissons également les extensions d'un processus de branchement $\beta = \langle B, E, F, l\rangle$. Ce sont les évènements possibles après des évènements de β mais non compris dans β, qui utilisent donc les conditions de β. Une extension e de β est encore une fois une paire $\langle t, {}^{\bullet}e\rangle$, où $t \in T$ (on notera $t = l(e)$) et ${}^{\bullet}e \subseteq B$ telle que :

- ${}^{\bullet}e$ est un co-set,
- la restriction de l à ${}^{\bullet}e$ est une bijection entre ${}^{\bullet}e$ et ${}^{\bullet}t$,
- $\nexists e' \in \beta$ t.q. $l(e') = t$ et ${}^{\bullet}e' = {}^{\bullet}e$.

On notera $PE(\beta)$ l'ensemble des extensions d'un processus de branchement β.

Les extensions d'un processus de branchement se distinguent de celles d'un processus car elles peuvent consommer l'ensemble des conditions de β, et ne se restreignent pas aux conditions finales. Ainsi, une extension de β peut être un évènement qui serait en conflit avec des évènements de β. Ces extensions sont un moyen de construire le dépliage, car en les ajoutant à un processus de branchement, on obtient à nouveau un processus de branchement. C'est ce qui est formulé par la proposition 6.1. Il faut pour cela rajouter l'extension et les conditions produites par le nouvel évènement.

Proposition 6.1 *Étant donné un processus de branchement $\beta = \langle B, E, F, l\rangle$, soit $e \in PE(\beta)$ une extension de β. On montre qu'il existe un processus de branchement $\beta' = \langle B', E', F', l'\rangle$ (on notera $\beta' = \beta \cup \{e\}$), strictement plus grand (au sens de la relation de préfixe) que β et tel que $E' = E \cup \{e\}$.*

Preuve 6.1 (Proposition 6.1) *On construit $\beta' = \langle B', E', F', l'\rangle$ en considérant un ensemble de conditions $e^{\bullet}$ tel que :*

– *l' étend l à e et $e^\bullet$, de sorte que $l'(e) = l(e)$ et la restriction de l' à $e^\bullet$ est une bijection entre $e^\bullet$ et $l(e)^\bullet$,*
– *$B' = B \cup e^\bullet$,*
– *$E' = E \cup \{e\}$.*
– *F' est une extension de F telle que $\forall b \in {}^\bullet e,\ F'(b, e) = 1$ et $\forall b \in e^\bullet,\ F'(e, b) = 1$.*

Montrons tout d'abord que $\mathcal{O}' = \langle B', E', F' \rangle$ est un réseau d'occurrences. $\mathcal{O} = \langle B, E, F \rangle$ est un réseau d'occurrences auquel on rajoute e et $e^\bullet$. Alors, $\forall b \in e^\bullet$, ${}^\bullet b = \{e\}$ donc $|{}^\bullet b| \leq 1$. $\mathcal{O}$ est acyclique, les arcs rajoutés dans $\mathcal{O}'$ vont de $\mathcal{O}$ vers e et de e vers $e^\bullet$, ce qui ne peut définir de cycle. $\mathcal{O}$ étant fini par précédence, $\mathcal{O}'$ hérite directement de cette propriété. Enfin, puisque ${}^\bullet e$ est un co-set*, e ne peut pas être en conflit avec lui-même, et les conditions de $e^\bullet$ non plus puisqu'elles ne possèdent qu'un seul arc d'entrée.*

Nous montrons ensuite que $\beta' = \langle O', l' \rangle$ est un processus de branchement. Par définition de l'extension e, l' restreint à ${}^\bullet e$ est une bijection entre ${}^\bullet e$ et ${}^\bullet l'(e)$; respectivement, par définition de l', l' restreint à $e^\bullet$ est une bijection entre $e^\bullet$ et $l'(e)^\bullet$.

Par construction, β est un préfixe de β', et puisque de plus $\nexists e' \in \beta$ t.q. $l'(e') = l'(e)$ et ${}^\bullet e' = {}^\bullet e$, l'évènement e est unique à β' qui est donc plus grand que β.

6.1.2.3 Préfixe fini complet du dépliage d'un réseau de Petri

L'algorithme proposé dans [McMillan 95] permet de vérifier un réseau de Petri en calculant un préfixe fini complet de son dépliage. Le dépliage est tronqué en déterminant des évènements « cut-off » après lesquels le dépliage peut être stoppé. Il faut pour cela comparer les configurations locales des évènements.

Un évènement du dépliage $\mathcal{U}(\mathcal{N})$ d'un réseau de Petri $\mathcal{N}$ est un *évènement « cut-off »* s'il existe un autre évènement e' dans le dépliage tel que :
– $l(\mathsf{Cut}\,([e'])) = l(\mathsf{Cut}\,([e]))$ (les marquages finaux sont identiques) ;
– $|[e']| < |[e]|$ (la configuration locale de e' est plus « courte » que celle de e).

L'algorithme de [McMillan 95] calcule un processus de branchement ne dépliant pas les évènements « cut-off ». Il est prouvé que ce processus de branchement est complet, c'est-à-dire que l'ensemble de ses coupures contient exactement les marquages accessibles. L'algorithme est amélioré par [Esparza 96] en proposant d'utiliser un autre ordre pour comparer les configurations locales des évènements. Cela permet d'obtenir un préfixe plus compact.

Exemple 6.1 *Dans le processus de branchement de la figure 6.2a, l'évènement e_4 est un « cut-off ». En effet, le marquage atteint après $[e_4]$ est $\{p_1, p_2\}$; or ce marquage est égal au marquage initial déjà produit par l'évènement $\bot$, et donc tel que $|[\bot]| < |[e_4]|$.*

On peut en conclure que le processus de branchement 6.2a est un préfixe fini complet du dépliage du réseau de Petri de la figure 6.1.

6.1.3 Processus temporels

Dans le cas des réseaux de Petri temporels, nous rappelons tout d'abord les travaux de Aura et Lilius [Aura 97], qui consistent à définir une sémantique d'ordres partiels des TPN à l'aide de processus temporels. Ils imposent pour cela certaines restrictions sur les TPN considérés : les réseaux utilisés doivent être saufs, et pour toute transition t on suppose que ${}^\bullet t > \mathbf{0}$ et $t^\bullet > \mathbf{0}$. Ces conditions sont dorénavant supposées vérifiées par tous les TPN considérés et ce, jusqu'à la fin de ce manuscrit.

Définition 6.5 (Processus temporel) *Soit $\mathcal{N} = \langle P, T, {}^\bullet(.), (.)^\bullet, M_0, I_s \rangle$ un réseau de Petri T-temporel et E' un processus dans le réseau de Petri sous-jacent. Un* processus temporel *est une paire $\langle E', \theta \rangle$, où $\theta : E' \to \mathbb{R}^+$ est une* fonction de temporisation *qui associe à chaque évènement de E' une* date de tir *qui doit être valide.*

Pour un co-set *$B' \subset E'^\bullet$ et une transition t sensibilisée par $l(B')$, on définit la date de sensibilisation dans B' de la transition t par :*

$$\mathsf{TOE}\,(B', t) = \max(\{\theta({}^\bullet b) \mid b \in B' \ \wedge \ l(b) \in {}^\bullet t\}).$$

Puisque l'on considère maintenant des réseaux saufs, on remarque que pour une transition t sensibilisée par un co-set *B', l'ensemble des conditions $\{b \in B' \mid l(b) \in {}^\bullet t\}$ est unique.*

On définit aussi l'ensemble des évènements précédant temporellement un évènement e :

$$\mathsf{Earlier}\,(e) = \{e' \in E' \mid \theta(e') < \theta(e)\}$$

et $C_e = \mathsf{Cut}\,(\mathsf{Earlier}\,(e))$ est la coupure (c'est-à-dire le marquage) qui précède le tir de e.

La fonction de temporisation d'un processus temporel est valide si $\theta(\bot) = 0$ et si les contraintes suivantes sont vérifiées :

$$\forall e \in E' \ (e \neq \bot), \ \theta(e) \geq \mathsf{TOE}\,({}^\bullet e, l(e)) + \mathsf{eft}\,(l(e)) \tag{6.1}$$

$$\forall e \in E' \ (e \neq \bot), \ \forall t \in \mathsf{enabled}\,(l(C_e))\,, \ \theta(e) \leq \mathsf{TOE}\,(C_e, t) + \mathsf{lft}\,(t) \tag{6.2}$$

Les auteurs font remarquer que les contraintes temporelles introduites dans les TPN rajoutent des contraintes de causalité supplémentaires entre les évènements, non représentées dans l'ordre partiel défini par un processus. C'est pour cette raison que la vérification de la validité d'une temporisation pour un évènement fait intervenir des contraintes sur l'ensemble des transitions sensibilisées par le marquage atteint en C_e.

Processus temporellement complet Un processus E' sera dit *temporellement complet* pour la temporisation θ, si pour toutes les extensions $e \in PE(E')$ la condition suivante est vérifiée :

$$\max\{\theta(e') \mid e' \in E'\} \leq \mathsf{TOE}\,({}^\bullet e, l(e)) + \mathsf{lft}\,(l(e)) \tag{6.3}$$

Un processus est donc temporellement complet si toutes les parties concurrentes du système ont été dépliées jusqu'à une même date temporelle, c'est-à-dire qu'il n'existe pas d'évènement concurrent devant être tiré avant la dernière date de ce processus. On remarque qu'un processus temporel valide est nécessairement temporellement complet.

Cette condition de complétude peut être vérifiée plus simplement en ne s'intéressant qu'aux conditions finales du processus. Si les conditions 6.1 de sensibilisation minimales sont vérifiées par θ pour tous les évènements, il n'est en effet nécessaire de s'assurer de cette condition que pour les « derniers » évènements de E', c'est-à-dire ceux dont les conditions produites ne sont pas consommées. De plus, il n'est pas nécessaire de vérifier cette condition si l'évènement produit des conditions consommées par l'extension. Dans ces conditions un processus est temporellement complet si pour toute extension $e \in PE(E')$:

$$\forall e' \in E', \ (e'^{\bullet\bullet} = \emptyset \ \wedge \ e' \notin {}^{\bullet\bullet} e) \ \Rightarrow \ (\theta(e') \leq \mathsf{TOE}\,({}^\bullet e, l(e)) + \mathsf{lft}\,(l(e))) \tag{6.4}$$

Entrelacements d'un processus temporel

Définition 6.6 (Entrelacement) *Un* entrelacement *des évènements d'un processus temporel $\langle E', \theta \rangle$ est un séquence finie ou infinie $\rho = e_1; e_2; e_3 \dots$ constituée des évènements de E', de sorte que tous les évènements apparaissent une et une seule fois. Elle définit un ordre total entre les évènements de E', figuré par l'indice i d'un évènement e_i qui correspond à sa position dans la séquence. Cet ordre total doit préserver à la fois l'ordre causal et l'ordre temporel, c'est-à-dire que :*

$$\forall i, j,\ (e_i < e_j \ \vee\ \theta(e_i) < \theta(e_j)) \Rightarrow i < j$$

Les entrelacements d'un processus temporel permettent de retrouver les exécutions séquentielles du réseau de Petri temporel, telles que définies par la sémantique séquentielle. On note $FS(\rho)$ l'exécution associée à un entrelacement ρ ; elle est définie par :

$$FS(\rho) = q_0 \xrightarrow{\theta(e_1)-\theta(\perp)} q_0{+}d_0 \xrightarrow{l(e_1)} q_1 \xrightarrow{\theta(e_2)-\theta(e_1)} q_1{+}d_1 \xrightarrow{l(e_2)} q_2 \xrightarrow{\theta(e_3)-\theta(e_2)} q_2{+}d_2 \xrightarrow{l(e_3)} q_3 \dots$$

Pour un entrelacement ρ et un entier k, on définit le préfixe ρ_k de longueur k constitué par les k premiers évènements de ρ, et on note E_k l'ensemble des évènements de ρ_k : $E_k = \{e_i \in E' \mid i \leq k\}$.

Exemple 6.2 *On donne un exemple de processus temporel sur la figure 6.3a, en associant des temporisations θ aux évènements du processus 6.2b. Cette temporisation est valide pour le réseau de Petri temporel de la figure 6.1, en effet :*

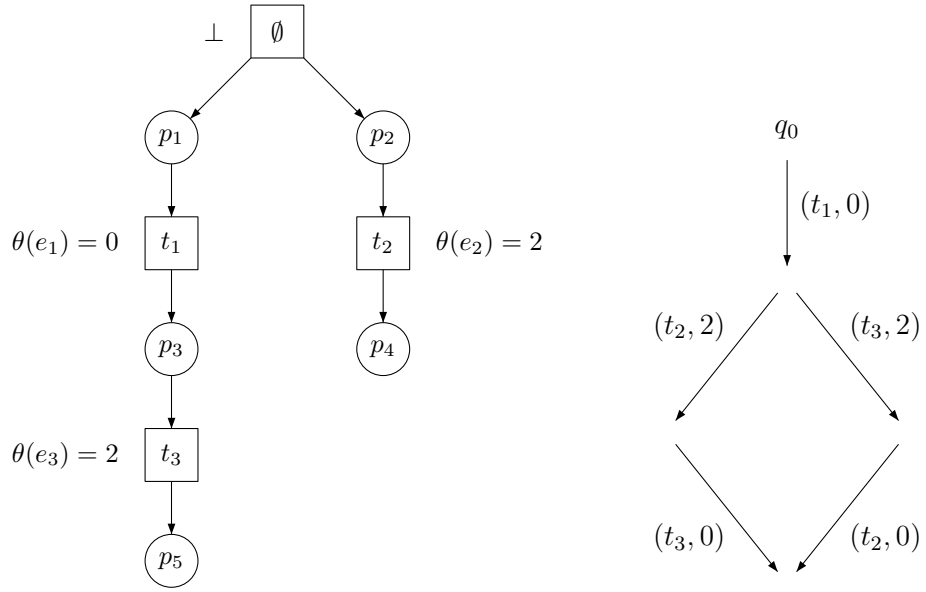
- *Pour l'évènement e_1, la date minimum de sensibilisation donnée par l'équation 6.1 est 0, or $\theta(e_1) \geq 0$. Les transitions sensibilisées avant e_1 sont t_1 et t_2 ; leur date de tir maximum donnée par l'équation 6.2 est respectivement ∞ et 2, or $\theta(e_1) \leq 2$.*
- *Pour l'évènement e_2, la date minimum de sensibilisation est 1, or $\theta(e_2) \geq 1$. Les transitions sensibilisées avant e_2 sont t_2 et t_3, elles doivent toutes deux être tirées avant la date 2, or $\theta(e_2) \leq 2$.*
- *Pour l'évènement e_3 la date minimum de sensibilisation est égale à $\theta(e_1) + 2 = 2$, or $\theta(e_3) \geq 2$. Les transitions sensibilisées avant e_3 sont t_2 et t_3, elles doivent être tirées avant la date 2, or $\theta(e_3) \leq 2$.*

On peut montrer que ces temporisations sont les seules valides pour ce processus : en effet, pour pouvoir tirer t_3 dès la première itération de la boucle il est obligatoire que t_1 et t_2 soient tirées respectivement en 0 et en 2.

Ce processus temporel admet deux entrelacements suivant le choix du premier évènement entre e_2 et e_3 puisqu'ils sont tirés à la même date. Ces entrelacements sont figurés sur la figure 6.3b.

6.1.4 Processus étendus et dépliage d'un TPN

Contrairement au cas non temporel, les processus temporels ne permettent pas de calculer un dépliage du réseau en examinant séparément les parties parallèles. En effet, dans le cas temporel des relations de causalités peuvent exister même entre des transitions appartenant à deux parties apparemment indépendantes. Ainsi, les processus temporels imposent de regarder l'état global du réseau. Malgré cela, Chatain et Jard ont développé une méthode permettant de construire un préfixe fini complet du dépliage d'un TPN [Chatain 06b]. Ils définissent



(a) Processus temporel valide

(b) Entrelacements

FIG. 6.3: Un processus temporel issu du réseau de Petri temporel 6.1, et ses entrelacements

pour cela une sémantique opérationnelle concurrente des TPN qui permet de déterminer la franchissabilité d'une transition en ne s'intéressant qu'à un marquage partiel du réseau.

Cette sémantique utilise une condition locale de tir LFC qui détermine si une transition t peut être franchie à une date θ, à partir d'un marquage partiel L :

- t doit être sensibilisée par L ;
- à la date θ, la durée minimale de sensibilisation est atteinte ;
- toute transition pouvant consommer des jetons de L, soit n'est pas sensibilisée, soit n'a pas dépassé à la date θ sa durée maximale de sensibilisation ;
- le marquage L est minimal.

Ainsi en pratique, pour déterminer le tir d'une transition, il faut s'intéresser aux jetons qu'elle consomme et éventuellement aux jetons pouvant sensibiliser une transition en conflit.

Cette sémantique produit des *processus étendus* qui complètent les processus temporels avec des arcs de lecture. Ces arcs relient une condition à un évènement ce qui rajoute une relation de causalité à l'évènement. Les auteurs prouvent que leur sémantique opérationnelle est correcte et complète en reliant chaque processus étendu produit par leur sémantique à un processus temporel, et respectivement. Ceci permet de définir le dépliage d'un TPN. On peut alors définir un dépliage symbolique en considérant non pas des valeurs temporelles fixées mais des dates de tir symboliques, contraintes par des systèmes d'inéquations.

6.1.4.1 Préfixe fini complet du dépliage d'un TPN

Les auteurs vont plus loin en définissant un préfixe fini complet du dépliage d'un TPN. Dans le cas non temporel, il suffisait de comparer les marquages : si deux processus produisent le même marquage ils ont le même futur. Ce n'est plus vrai avec le temps puisqu'il est aussi nécessaire de comparer les dates de tir. Cependant ces dates ne font que progresser avec le temps ; les auteurs définissent donc une condition pour comparer des processus étendus

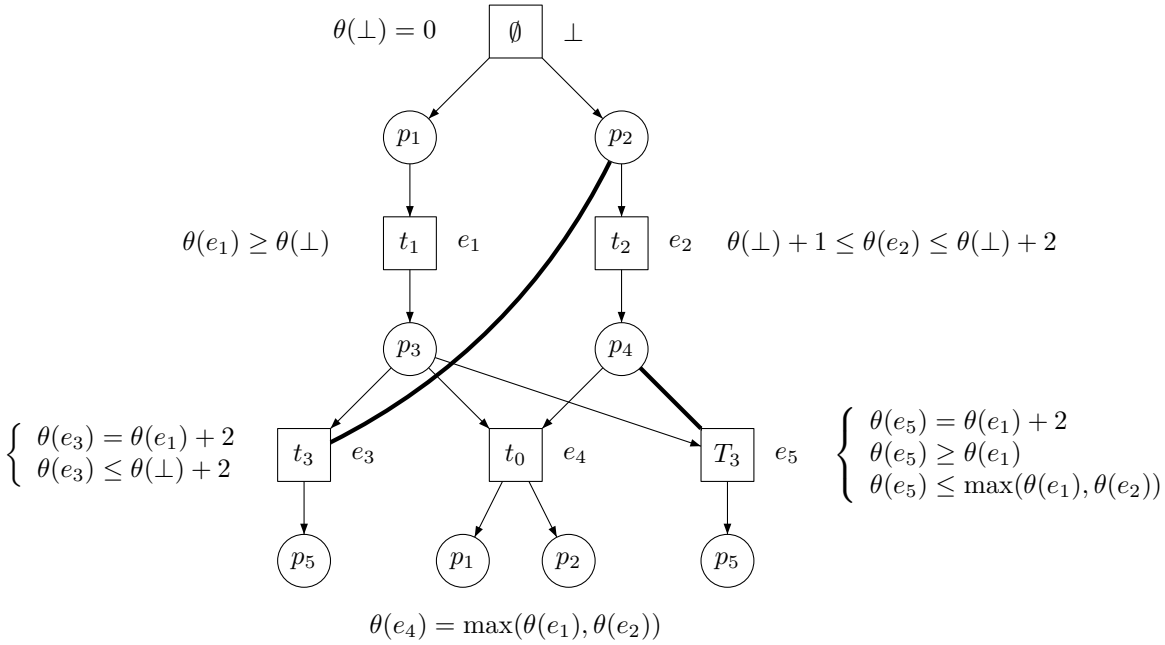


FIG. 6.4: Préfixe fini complet du dépliage du TPN 6.1 [Chatain 06b]

en comparant les âges des conditions finales. En utilisant cette condition ils déterminent un préfixe du dépliage. Ce préfixe est complet puisque par substitution il est possible de reconstruire le dépliage entier.

Exemple 6.3 *Un préfixe fini complet du dépliage du TPN de la figure 6.1 est présenté sur la figure 6.4. Les arcs sans flèches sont les arcs de lecture. Les contraintes sur les dates de tir sont écrites à côté des évènements.*

On s'aperçoit que le tir de t_3 est dupliqué pour distinguer le tir possible avant e_2, lorsqu'un jeton en p_2 est présent, et celui possible après e_2 lorsque p_4 est marquée.

6.1.5 Analyse de scénarios en logique linéaire

La méthode d'analyse par la logique linéaire des réseaux de Petri temporels paramétrés en sémantique forte introduite dans [Delfieu 07] permet de synthétiser des contraintes sur les paramètres afin de vérifier un scénario spécifié. Un *scénario de tir* dans un réseau de Petri est défini comme un multi-ensemble sur l'ensemble T des transitions du réseau. La procédure décrite dans [Girault 97] permet de vérifier l'accessibilité de ce scénario par une seule preuve d'une formule de logique linéaire, ceci quels que soient les entrelacements possibles entre les transitions du scénario qui ne sont alors pas calculés.

La traduction des réseaux de Petri en logique linéaire utilise deux opérateur $\otimes$ (*fois*) et $\multimap$ (*entraîne*). L'opérateur $\otimes$ exprime la conjonction de ressources et permet de représenter les marquages ; $\multimap$ exprime la transformation d'une ressource, c'est-à-dire le tir d'une transition.

Exemple 6.4 *Le réseau de Petri non temporel correspondant à la figure 6.1 est traduit par*

les formules de logique linéaire suivantes :

$$\begin{array}{ll} M_0 : & p_1 \otimes p_2 \\ t_1 : & p_1 \multimap p_3 \\ t_2 : & p_2 \multimap p_4 \\ t_3 : & p_3 \multimap p_5 \\ t_0 : & p_3 \otimes p_4 \multimap p_1 \otimes p_2 \end{array}$$

À partir de la transformation du réseau sous la forme d'un ensemble de formules de logique linéaire, l'analyse d'un scénario est basée sur la théorie du calcul des séquents. On détermine dans un premier temps un séquent, c'est-à-dire une proposition à démontrer ; ce séquent a la forme suivante : $p_1 \otimes p_2 \ldots \otimes p_n, t_1, t_2 \ldots, t_n \vdash C$. Les formules p_i constituent le marquage initial, $t_1, t_2 \ldots, t_n$ le scénario à analyser et C est la ressource conclusion qui doit correspondre au marquage final.

La méthode de preuve applique des règles d'élimination des connecteurs logiques de la formule, ce qui revient à tirer successivement des transitions du scénario. Temporellement, on peut au cours de la preuve calculer symboliquement les dates de tir des transitions. Une première méthode d'analyse [Pradin-Chézalviel 99] était limitée à la sémantique faible des TPN dans laquelle les contraintes temporelles n'influencent pas les relations causales. Au contraire, en sémantique forte, on doit forcer le tir d'une transition si son horloge atteint la valeur maximale de la durée de sensibilisation. Nous avons donc adaptée la méthode à la sémantique forte [Delfieu 07].

Groupes de conflits

Nous l'avons déjà évoqué pour construire les processus temporels, dans les TPN en sémantique forte les contraintes temporelles ajoutent des causalités supplémentaires entre des évènements *a priori* concurrents. En effet, considérons le réseau de Petri temporel présenté sur la figure 6.5. Pour pouvoir tirer t_1 il est nécessaire que la transition t_2 ait été désensibilisée, et donc que t_3 ait été tirée préalablement. Il existe donc une dépendance causale entre le tir de t_1 et celui de t_3. Nous notons en particulier qu'à cause de cette dépendance le marquage $\{p_4, p_7\}$ n'est pas accessible.

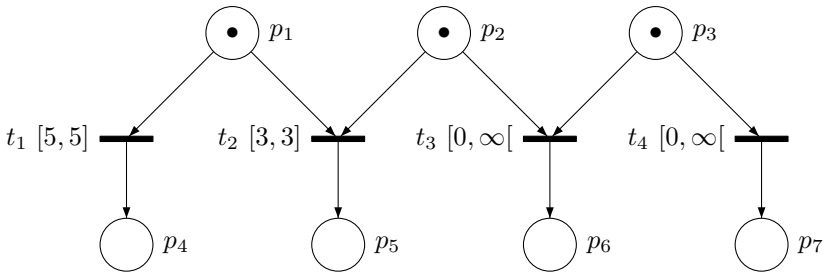


FIG. 6.5: Groupe de conflits

Pour prendre en compte ces dépendances, dans la méthode d'analyse de scénario en sémantique forte, on calcule des groupes de conflits en regroupant transitivement les transitions en conflit structurel. Dans l'exemple précédent, les 4 transitions t_1, t_2, t_3, t_4 sont regroupées dans le même groupe de conflits. Lors de la preuve d'un scénario, un ordre total est alors fixé

entre les évènements d'un groupe de conflit ; il est donc nécessaire de réintroduire des entrelacements, en effectuant plusieurs preuves, pour obtenir l'ensemble des dates de tir possibles.

L'analyse temporelle qui est ensuite effectuée prend en compte les groupes de conflits calculés, mais elle sort finalement du cadre de la logique linéaire. Dans la méthode de dépliage que nous présentons dans la partie suivante, nous reprenons et nous enrichissons l'analyse des conflits effectuée. La détermination des dates de tirs des évènements qui en découle ne sera donc présentée que dans le cadre des dépliages.

6.2 Dépliage de réseaux de Petri temporels paramétrés

À partir des travaux existants, nous proposons une nouvelle méthode de dépliage. Nous souhaitons définir un dépliage plus compact que celui proposé par Chatain et Jard. Leur méthode de dépliage présente l'intérêt de proposer une condition locale pour déterminer de manière certaine le tir d'une transition. Cela impose par contre de dupliquer de nombreuses fois certains évènements pour distinguer les conditions dans lesquelles le tir intervient. À l'inverse, dans l'approche des réseaux de Petri par la logique linéaire, l'analyse de scénarios de tirs que nous avions proposée ne devait dupliquer les évènements que dans les cas de conflits. Les limitations de cette méthode résidaient dans la nécessité de déterminer des scénarios à analyser.

Nous souhaitons donc reprendre la méthode d'analyse des conflits proposée dans l'approche par la logique linéaire, en l'associant à une procédure de construction du dépliage. Le principe est d'étudier les situations de conflits lorsqu'un choix non déterministe entre plusieurs évènements est possible. Cela permet de déduire les dates de tirs possibles pour les évènements, en fonction de celles des évènements précédents ; ces dates de tirs sont alors représentées par un domaine symbolique.

Structurellement le dépliage d'un réseau de Petri temporel paramétré est identique à celui du réseau de Petri sous-jacent. Cependant pour associer des dates de tirs aux évènements il pourra être nécessaire de distinguer les conditions dans lesquelles sont tirés les évènements. Cela revient alors à dupliquer temporellement les évènements, mais ces duplications seront limitées aux situations de conflits et correspondent donc au degré d'indéterminisme dans le passé des évènements.

Le résultat est un dépliage plus compact. L'inconvénient est qu'il n'est plus possible de déterminer des conditions suffisantes pour vérifier la validité temporelle d'un évènement. On applique donc une approche optimiste : on ajoute l'évènement au dépliage si les contraintes symboliques connues sur la date de tir de l'évènement l'autorisent ; mais on se doit alors de recalculer les dates de tir possible si ces contraintes sont modifiées.

6.2.1 Analyse des conflits

Dans les réseaux d'occurrences il existe déjà une relation de conflit entre deux nœuds. Cette relation décrit que deux nœuds du réseau sont séparés causalement par un choix non déterministe amont. Ce sont ces situations de choix que nous allons analyser en détail afin de déterminer les dates de tirs des évènements. Nous définissons pour cela une nouvelle relation de *conflit direct* qui ne s'applique cette fois qu'entre deux évènements, au moment même du choix non déterministe. Une seconde relation, dite de *libération*, utilise cette nouvelle relation de conflit afin de représenter des dépendances supplémentaires, telles que celles présentes dans le réseau de la figure 6.5.

Conflits directs Dans un réseau d'occurrences, nous définissons que deux évènements e_1, e_2 sont en *conflit direct*, ce que l'on note e_1 conf e_2, s'ils partagent une condition d'entrée, mais qu'aucune de leurs préconditions n'est en conflit, c'est-à-dire que :

$$e_1 \text{ conf } e_2 \iff \begin{cases} {}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset, \\ \forall b \in {}^\bullet e_1, \neg(b \# e_2), \\ \forall b \in {}^\bullet e_2, \neg(b \# e_1) \end{cases}$$

Les deux dernières conditions équivalent à dire que les nœuds de ${}^\bullet e_1 \cup {}^\bullet e_2$ forment un co-set.

Si les préconditions des évènements sont en conflit cela signifie que le choix a déjà été déterminé en amont dans le passé causal. On peut ainsi démontrer le lemme 6.2 qui spécifie que l'existence d'un conflit entre deux évènements, selon la relation de conflit définie entre les nœuds d'un réseau, implique nécessairement l'existence en amont d'un couple d'évènements en conflit direct.

Lemme 6.2 *Soit e, e' deux évènements dans un réseau d'occurrences $\mathcal{O} = \langle B, E, F \rangle$.*

$$e \# e' \implies \exists e_1 \leq e \text{ et } \exists e'_1 \leq e' \text{ t.q. } e_1 \text{ conf } e'_1$$

Preuve 6.2 (Lemme 6.2) *Si $e \# e'$ alors il existe deux chemins $s_i e_i \dots e$ et $s_i e'_i \dots e'$, avec $s_i \in B$, $e_i, e'_i \in E$ et $e_i \neq e'_i$. e_i et e'_i partagent donc une condition d'entrée commune s_i.*
Soit on obtient directement e_i conf e'_i, sinon cela signifie qu'une des préconditions $s_{i-1} \in {}^\bullet e_i$ (autre que s_i) de e_i (ou de e'_i, ce cas se démontrerait de manière similaire) est en conflit avec e'_i : $s_{i-1} \# e'_i$. On en déduit de nouveau qu'il existe deux chemins $s_j e_j \dots s_{i-1}$ et $s_j e'_j \dots e'_i$, avec $s_j \in B$, $e_j, e'_j \in E$ et $e_j \neq e'_j$. On les concatène aux chemins précédents, ce qui donne à nouveau deux chemins : $s_j e_j \dots e$ et $s_j e'_j \dots e'$.
Par induction inverse, puisque que $\mathcal{O}$ est fini par précédence, il existe nécessairement deux chemins $s_n e_n \dots e$ et $s_n e'_n \dots e'$, avec $s_n \in B$, $e_n, e'_n \in E$ et $e_n \neq e'_n$, tels que $\forall x \in {}^\bullet e_n, \neg(x \# e'_n)$ et $\forall x \in {}^\bullet e'_n, \neg(x \# e_n)$, ce qui prouve que e_n conf e'_n, et donc le lemme.

Évènements libérateurs Dans l'exemple de la figure 6.5 on remarquait que dans des situations où plusieurs conflits se trouvaient reliés entre eux, des conflits effectifs pouvaient être désactivés par le tir préalable d'une autre transition, entraînant ainsi des relations de causalités supplémentaires.

Nous proposons de caractériser ces situations par une relation ternaire, dite de *libération* (notée lib), qui regroupe par transitivité deux conflits directs, c'est-à-dire qu'un évènement est en conflit direct avec deux autres évènements :

$$(e_1, e_2, e_3) \in \text{lib} \iff \begin{cases} e_1 \text{ co } e_3, \\ e_1 \text{ conf } e_2 \text{ et } e_2 \text{ conf } e_3 \end{cases}$$

Grâce à cette relation, on note que le tir de e_1 peut libérer e_3 du conflit avec e_2 (symétriquement on a $(e_3, e_2, e_1) \in$ lib). On notera aussi de manière binaire e lib e'' s'il existe e' tel que $(e, e', e'') \in$ lib.

Exemple 6.5 *La figure 6.6 présente plusieurs réseaux d'occurrences illustrant différentes situations de conflits :*

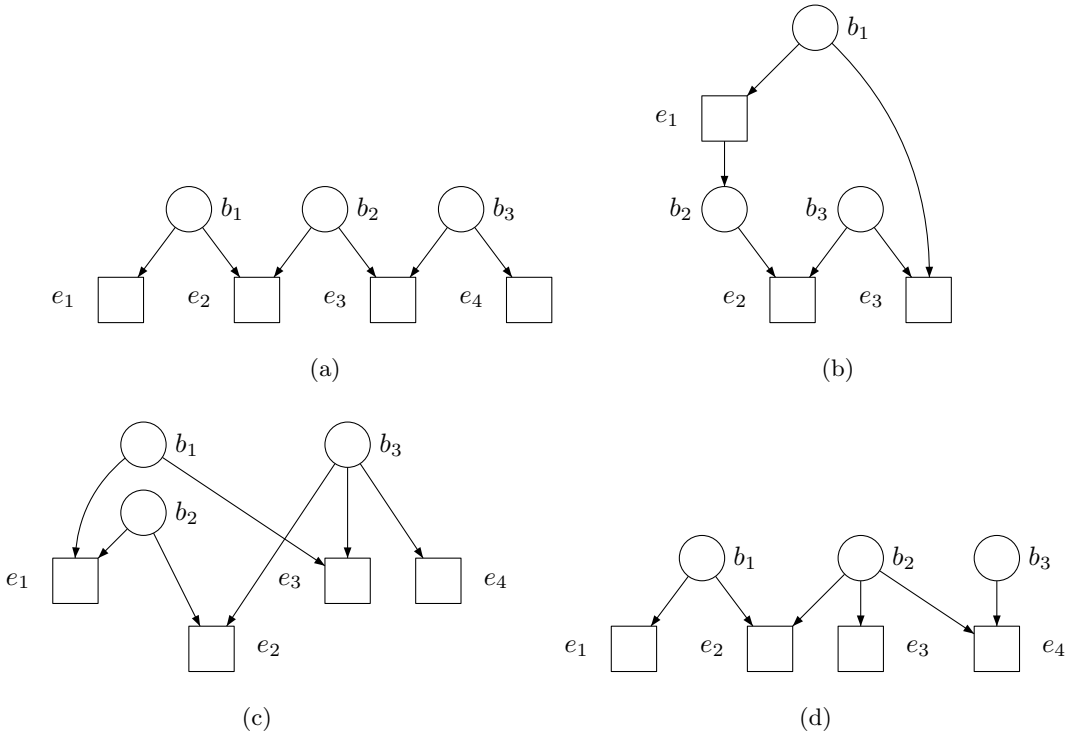


FIG. 6.6: Conflits dans des réseaux d'occurrences

- *Dans le réseau de la figure 6.6a (qui correspond en partie au dépliage du réseau de Petri de la figure 6.5), on a les relations de conflits suivantes :* e_1 conf e_2, e_2 conf e_3 *et* e_3 conf e_4. *Cela permet de déduire les relations de libérations :* $(e_1, e_2, e_3) \in$ lib *et* $(e_2, e_3, e_4) \in$ lib .
- *Dans le réseau de la figure 6.6b, la relation* e_1 conf e_3 *est vérifiée grâce à la condition commune* b_1. *Par contre il n'y a pas* e_2 conf e_3, *malgré la condition commune* b_3, *car la condition* b_2 *consommée par* e_2 *est en conflit avec* e_3 : $b_2 \# e_3$ *(ou encore* $\{b_1, b_2, b_3\}$ *n'est pas un* co-set*). Dans cet exemple le conflit direct entre* e_1 *et* e_3 *détermine le conflit entre* e_2 *et* e_3.
- *Dans le réseau de la figure 6.6c, on illustre qu'un évènement peut libérer deux conflits. On a les relations de conflits :* e_1 conf e_2, e_1 conf e_3, e_2 conf e_4 *et* e_3 conf e_4 *; et donc les relations de libérations suivantes :* $(e_1, e_2, e_4) \in$ lib *et* $(e_1, e_3, e_4) \in$ lib , *où l'on s'aperçoit que* e_4 *libère* e_1 *des conflits respectifs qu'il a avec* e_2 *et* e_3 *(et réciproquement).*
- *Enfin dans le réseau de la figure 6.6d, on illustre cette fois qu'un conflit peut être libéré par plusieurs évènements libérateurs :* e_1 conf e_2, e_2 conf e_3, e_2 conf e_4 *et* e_3 conf e_4 *; et donc* $(e_1, e_2, e_3) \in$ lib *et* $(e_1, e_2, e_4) \in$ lib . *Ainsi* e_1 *peut être libéré du conflit avec* e_2 *soit par* e_3, *soit par* e_4. *Réciproquement,* e_1 *libère à la fois* e_3 *et* e_4 *du conflit avec* e_2, *par contre le conflit entre* e_3 *et* e_4 *ne peut pas être libéré.*

6.2.2 Processus de branchement temporels

Nous proposons maintenant de déplier des réseaux de Petri temporels paramétrés en définissant des processus de branchement temporels. On ajoute pour cela une fonction de tem-

porisation, de la même manière que les processus temporels étendent la notion de processus. Mais contrairement à un processus, un processus de branchement possède des évènements en conflit. En conséquence une fonction de temporisation va associer des dates finies à un ensemble d'évènements sans conflit du processus de branchement (ces évènements forment alors un processus), et au contraire, une valeur infinie aux autres évènements.

Cependant, nous avons vu sur le réseau de Petri temporel de la figure 6.5 que l'introduction du temps modifie les relations de causalités qui existent normalement entre les évènements d'un réseau de Petri. Dans cet exemple, l'évènement correspondant au tir de t_1 nécessite le tir au préalable de la transition t_3, bien que ces deux transitions ne soient pas directement reliées entre elles. Pour déplier un réseau de Petri temporel nous devons donc représenter cette relation de causalité. Dans [Chatain 06b] cela est réalisé en rajoutant un arc de lecture depuis une condition suivant le tir de t_3, mais cela amène à dupliquer l'évènement pour chacune des conditions possibles.

Dans notre approche cette situation est mise en valeur par la relation de libération. On est alors amené à considérer deux cas pour calculer la date de tir d'un évènement, en distinguant le tir de l'évènement dans une situation de conflit, ou bien le tir de l'évènement après que le conflit a été libéré par un évènement précédent.

Définition 6.7 (Processus de branchement temporel) *Étant donné un réseau de Petri temporel paramétré* $\mathcal{N} = \langle P, T, Par, {}^\bullet(.), (.)^\bullet, M_0, J_s, D_p \rangle$ *et une valuation* $\nu \in D_p$ *des paramètres, un* un processus de branchement temporel *de* $\mathcal{N}$ *est un triplet* $\langle \beta, \nu, \theta \rangle$ *qui associe à un processus de branchement* $\beta = \langle B, E, F, l \rangle$*, une fonction de temporisation* $\theta : E \to \mathbb{R}^+ \cup \{\infty\}$ *qui doit être valide.*

θ associe à chaque évènement de β une date de tir $\theta(e)$ *qui est une date soit finie, soit égale à l'infini. Elle sera valide si et seulement si* $\theta(\bot) = 0$*, et* $\forall e \in E,\ e \neq \bot$*, les valeurs* $\theta(e)$ *des dates de tir, avec les valeur des paramètres définies par* ν*, doivent vérifier le système de contraintes suivant :*

$$\Big[\theta(e) \neq \infty \ \wedge\ \theta(e) \geq \mathsf{TOE}\,({}^\bullet e, l(e)) + \mathsf{eft}\,(l(e)) \tag{6.5}$$

$$\wedge\quad \theta(e) \leq \mathsf{TOE}\,({}^\bullet e, l(e)) + \mathsf{lft}\,(l(e)) \tag{6.6}$$

$$\wedge\quad \Big[\forall e' \in E \ \textit{t.q.}\ e' \ \mathsf{conf}\ e,\ \theta(e') = \infty \wedge$$

$$[\theta(e) \leq \mathsf{TOE}\,({}^\bullet e', l(e')) + \mathsf{lft}\,(l(e'))\,] \tag{6.7}$$

$$\vee\ [\exists e'' \ \textit{t.q.}\ (e, e', e'') \in\ \mathsf{lib}\ \wedge\ \theta(e'') < \theta(e)]\Big]\Big] \tag{6.8}$$

$$\vee\Big[\theta(e) = \infty\ \wedge\ \exists b \in {}^\bullet e,\ \theta({}^\bullet b) = \infty\Big] \tag{6.9}$$

$$\vee\Big[\theta(e) = \infty\ \wedge\ \exists e' \in E \ \textit{t.q.}\ e' \ \mathsf{conf}\ e\ \wedge\ \theta(e') \neq \infty\Big] \tag{6.10}$$

Dans ces contraintes en conjonction ou en disjonction, les opérateurs max, + *et les opérateurs de comparaison sont étendus sur* $\mathbb{R}^+ \cup \{\infty\}$*. Les paramètres apparaissent dans ce système dans les expressions linéaires* eft *et* lft*.*

L'inéquation 6.5 spécifie la date minimum de tir, qui dépend des conditions consommées par le tir et de la durée minimale de sensibilisation $\mathsf{eft}\,(l(e))$ de la transition $l(e)$ (durée éventuellement paramétrée).

De manière similaire, l'inéquation 6.6 spécifie la date maximum de tir, due à la durée maximale de sensibilisation $\mathsf{lft}(l(e))$. Cette durée peut également être paramétrée, mais peut aussi être égale à l'infini, ce qui rendrait alors immédiatement la contrainte tautologique.

Enfin, si l'on souhaite assigner une date finie à un évènement e, les évènements qui sont en conflit avec lui doivent nécessairement posséder une date de tir infinie ; ainsi pour tout conflit, un seul des évènements pourra avoir une date finie, ce qui va permettre à θ de définir un processus temporel. Il est ensuite nécessaire de vérifier qu'aucun évènement en conflit avec e ne doit être tiré avant lui. Pour cela on doit vérifier pour chaque conflit direct, soit l'inéquation 6.7 qui contraint la date de tir de e à être inférieure à la date maximum de tir de l'évènement en conflit ; soit l'inéquation 6.8 qui spécifie que le conflit a été désactivé par le tir d'un évènement libérateur précédant strictement e. Grâce à cette dernière contrainte stricte on s'assure que deux évènements libérateurs ne se libèrent pas mutuellement d'un même conflit. Elle permet alors de démontrer le lemme 6.3 qui spécifie que pour tout conflit d'un évènement e avec un évènement e' il existe un conflit e' conf e'' qui est effectif, c'est dire sans libérateur.

Lemme 6.3 *Dans un processus de branchement temporel* $\langle B, E, F, l, \nu, \theta\rangle$ *:*

$$(e \ \mathsf{conf}\ e' \ \wedge\ \theta(e) \neq \infty) \implies (\exists e''\ \textit{t.q.}\ e''\ \mathsf{conf}\ e'\ \textit{et}\ \theta(e'') \leq \mathsf{TOE}\,({}^\bullet e', l(e')) + \mathsf{lft}\,(l(e'))$$

Preuve 6.3 (Lemme 6.3) *Si e vérifie l'inéquation 6.7, le lemme est directement démontré par $e'' = e$.*
Sinon, e est libéré de son conflit avec e' en vérifiant l'inéquation 6.8 : dans ce cas $\exists e_1$ t.q. $(e, e', e_1) \in \mathsf{lib}$, et donc e_1 conf e'. A son tour e_1 vérifie soit 6.7 et le lemme est démontré, soit 6.8.
Dans ce dernier cas il existe un nouvel évènement e_2 qui libère cette fois-ci e_1 du conflit avec e', i.e. $(e_1, e', e_2) \in \mathsf{lib}$: e_2 est nécessairement différent de e, car e étant libéré par e_1, la contrainte $\theta(e_1) < \theta(e)$ est vérifiée, or il faut maintenant vérifier $\theta(e_2) < \theta(e_1)$. On recommence ainsi jusqu'à trouver un évènement qui ne peut être libéré du conflit avec e' : cet évènement existe car à chaque étape les inéquations suivantes sont vérifiées : $\theta(e_1) < \theta(e)$, puis $\theta(e_2) < \theta(e_1)$, etc ... Or puisque le temps est supposé diverger, il ne peut y avoir une suite infinie d'évènements sans écoulement du temps. L'évènement qui ne sera pas libéré du conflit vérifie alors l'inéquation 6.7 pour le conflit avec e' ce qui démontre le lemme.

Alors que les quatre inéquations précédentes spécifient la date finie d'un évènement, les équations suivantes, ajoutées en disjonction, spécifient dans quels cas la date de l'évènement peut être infinie.

Premièrement, si l'une des préconditions de e ($b \in {}^\bullet e$) possède une date de production infinie, alors la date de tir de e est infinie : équations 6.9.

Deuxièmement, les équations 6.10 autorisent spécifiquement la date de tir de e à être infinie dans le cas où il existe un autre évènement en conflit dont la date de tir est finie. C'est la seule possibilité d'introduire des dates infinies dans le système d'équations. Ces dates infinies sont ensuite propagées par les équations 6.9.

On montre enfin que dans un processus de branchement temporel, l'ordre temporel est conforme à l'ordre causal, ce qui est exprimé par le lemme 6.4.

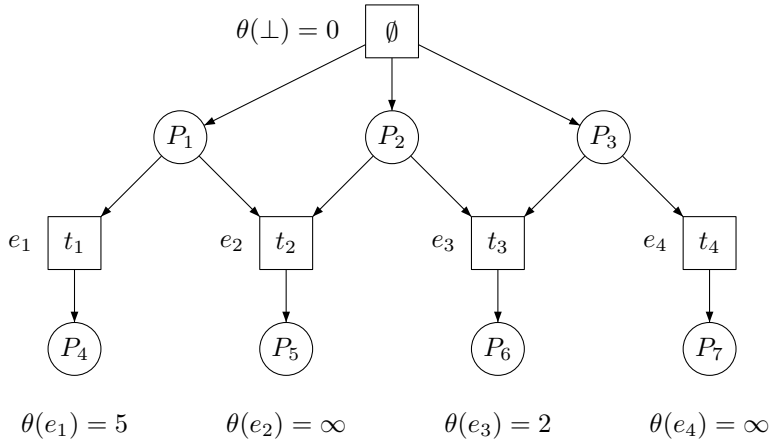


FIG. 6.7: Processus de branchement temporel issu du TPN de la figure 6.5

Lemme 6.4 *Dans un processus de branchement temporel* $\langle B, E, F, l, \nu, \theta \rangle$, $\forall e, e' \in E$:

$$e < e' \implies \theta(e) \leq \theta(e')$$

Preuve 6.4 (Lemme 6.4) *Si* $e < e'$ *alors il existe un chemin de* e *à* e' : $e_1 e_2 \dots e_{n-1} e_n$ *(avec* $e_1 = e$ *et* $e_n = e'$*). Dans ce chemin, pour* $1 \leq i \leq n-1$, $e_i < e_{i+1}$ *: si* $\theta(e_{i+1}) \neq \infty$ $\exists b_i \in e_i^\bullet \cap {}^\bullet e_{i+1}$ *et alors l'inéquation 6.5 implique que* $\theta(e_{i+1}) \geq \theta(e_i) + \underbrace{\mathsf{eft}\,(l(e_{i+1}))}_{\geq 0} \geq \theta(e_i)$ *;*
sinon $\theta(e_{i+1}) = \infty$ *et donc dans tous les cas* $\theta(e_i) \leq \theta(e_{i+1})$.
De proche en proche, on vérifie alors que $\theta(e_1) \leq \theta(e_n)$*, c'est-à-dire* $\theta(e) \leq \theta(e')$.

Exemple 6.6 *La figure 6.7 présente un processus de branchement temporel pour le réseau de Petri temporel de la figure 6.5. Dans ce processus de branchement temporel le tir de* t_1 *par* e_1 *est possible grâce au tir précédant de* t_3 *par* e_3 *qui libère* e_1 *du conflit avec* e_2*. Ainsi* e_1 *peut vérifier l'inéquation 6.8 ; sans cela* e_1 *ne vérifie pas l'inéquation 6.7 et ne peut donc pas posséder de date de tir finie sans le tir au préalable de* t_3.

6.2.2.1 Processus de branchement temporellement complet

Un processus de branchement $\beta = \langle E, B, F, l \rangle$ sera dit *temporellement complet* pour une valuation ν des paramètres et pour la temporisation θ, si pour toutes ses extensions $e \in PE(\beta)$:

$$\max\{\theta(e') \mid e' \in E \wedge \theta(e') \neq \infty\} \leq \mathsf{TOE}\,({}^\bullet e, l(e)) + \mathsf{lft}\,(l(e)) \tag{6.11}$$

Contrairement aux processus temporels, la temporisation d'un processus de branchement temporel définie par les contraintes de la définition 6.7 ne permet pas nécessairement d'obtenir des exécutions correctes. En effet, un processus de branchement temporel est *a priori* incomplet, dans le sens où toutes les contraintes temporelles ne sont pas connues, puisque tous les conflits ne sont pas inclus. La contrainte 6.11 permet dans ce cas de spécifier un processus de branchement temporel fini, mais en s'assurant que tous les évènements sont corrects.

Elle spécifie pour cela que la date de tir de tous les évènements est inférieure à la date de tir maximale de toutes les extensions possibles de β. Or les conflits qui n'ont pas encore

été découverts font partie de ces extensions ; ainsi les évènements de β ne seront possibles qu'avant ces conflits potentiels. De plus cela assure, comme pour les processus temporels, que le dépliage a été effectué parallèlement jusqu'à une même date dans toutes les branches parallèles.

Comme pour les processus temporels, cette contrainte peut être simplifiée si les contraintes 6.5 sont vérifiées, en ne s'intéressant qu'aux « derniers » évènements de β. La nouvelle condition est alors $\forall e \in PE(\beta)$:

$$\forall e' \in \beta,\ (\theta(e') \neq \infty\ \wedge\ e'^{\bullet\bullet} = \emptyset\ \wedge\ e' \notin {}^{\bullet\bullet}e) \implies \theta(e') \leq \mathsf{TOE}\,({}^{\bullet}e, l(e)) + \mathsf{lft}\,(l(e)) \quad (6.12)$$

Exemple 6.7 *Sur l'exemple de la figure 6.7, le processus de branchement temporel est nécessairement complet puisqu'il ne possède pas d'extensions.*

Par contre, un processus de branchement temporel valide pour le TPN de la figure 6.5 peut aussi comporter uniquement le tir de t_1 par e_1 à la date 5. Il ne serait cependant pas complet puisque e_2 est alors une extension possible dont la date de tir maximum est 3.

6.2.2.2 Extensions d'un processus de branchement temporel

En ajoutant les extensions au fur et à mesure à un processus de branchement temporel on peut construire par induction le dépliage d'un réseau de Petri temporel.

Proposition 6.5 *Étant donné un processus de branchement temporel $\Sigma = \langle \beta, \nu, \theta \rangle$ temporellement complet, avec $\beta = \langle B, E, F, l \rangle$, soit $e \in PE(\beta)$ une extension de β. On montre qu'il existe un processus de branchement temporel $\Sigma' = \langle \beta', \nu, \theta' \rangle$ tel que $\beta' = \beta \cup \{e\}$ et θ' est une extension de θ.*

Preuve 6.5 (Proposition 6.5) *On a déjà montré dans la proposition 6.1 que l'ajout d'une extension à un processus de branchement étendu permettait de créer un autre processus de branchement $\beta' = \beta \cup \{e\}$. Nous devons maintenant montrer qu'il existe une date de tir possible pour cette extension, sans modifier les dates de tir des évènements déjà présents.*

- *Si $\exists b \in {}^{\bullet}e$, t.q. $\theta({}^{\bullet}b) = \infty$, alors il suffit de prendre pour valeur $\theta'(e) = \infty$. Cela permet de vérifier l'équation 6.9. Les autres évènements ne sont pas modifiés : en effet même si e peut rajouter des conflits dans β', puisque sa date de tir maximum est infinie les contraintes 6.7 pour ces évènements sont toujours vérifiées.*
- *Si $\exists e' \in E$, t.q. e' conf e et $\theta(e') \neq \infty$, alors il suffit encore une fois de prendre pour valeur $\theta'(e) = \infty$, ce qui permet de vérifier l'équation 6.10. Puisque le processus de branchement temporel Σ est temporellement complet, les évènements en conflit vérifiaient la contrainte de complétude 6.11 sur la date de tir maximum de e. Ils vérifient donc toujours les inéquations 6.7.*
- *Enfin, si aucune de ces conditions n'est vérifiée, il faut trouver une valeur finie à $\theta(e)$. Si e n'introduit aucun conflit supplémentaire n'importe quelle valeur de $\theta'(e)$ vérifiant les inéquations 6.5 et 6.6 est satisfaisante.*
  Sinon, pour chaque conflit tel que $\exists e'$, e conf e', et tel que $\theta(e') = \infty$, cela peut signifier tout d'abord que e' vérifie l'équation 6.9 : dans ce cas la date maximum de tir pour e' est infinie, et donc e vérifie trivialement l'inéquation 6.7 pour son conflit avec e' ; sinon, cela signifie qu'il existe e'' tel que e'' conf e' et $\theta(e'') \neq \infty$ (équation 6.10). e'' et e ne peuvent alors pas être en relation causale car e conf e' entraîne que ${}^{\bullet}e' \cup {}^{\bullet}e$ est un co-set ce qui n'est plus le cas si $e'' < e$; e'' et e ne peuvent pas non plus être en conflit car alors l'une

des préconditions de e ou de e'' possède nécessairement une date de production infinie. Donc e'' co e, et en conséquence $(e, e', e'') \in$ lib . Il existe alors une valeur possible pour $\theta'(e)$ inférieur à sa date maximale $\mathsf{TOE}(^\bullet e, l(e)) + \mathsf{lft}(l(e))$, car le processus de branchement étant supposé temporellement complet, $\theta(e'') \leq \mathsf{TOE}(^\bullet e, l(e)) + \mathsf{lft}(l(e))$. Cela permet de vérifier l'inéquation 6.8 pour le conflit avec e'.
On choisit au final pour e la date de tir satisfaisant aux contraintes définies par chacun de ses conflits (dans le pire cas la seule date possible correspond à $\mathsf{TOE}(^\bullet e, l(e)) + \mathsf{lft}(l(e))$).

Nous notons que le processus de branchement temporel obtenu Σ' n'est pas nécessairement temporellement complet. En effet, seuls les conflits présents dans β' sont pris en compte ; or l'extension e pourrait être impossible à cause de conflits qui n'ont pas encore été ajoutés.

Toutefois, en posant des restrictions sur l'ordre d'ajout des extensions à un processus de branchement, on peut obtenir à nouveau un processus de branchement temporellement complet. En effet, si l'on choisit d'ajouter l'extension e possédant la date de tir maximum (calculée par l'expression $\mathsf{TOE}(^\bullet e, l(e)) + \mathsf{lft}(l(e))$) la plus faible, le processus de branchement temporel obtenu est nécessairement temporellement complet. Étant donnée une extension e de β possédant la date de tir maximum la plus faible, on montre que le nouveau processus de branchement temporel obtenu Σ' vérifie que la condition 6.11 de complétude :

- Pour toutes les extensions de β' qui sont également des extensions de β, la condition 6.11 est héritée de Σ, qui est supposé temporellement complet, et elle est donc vérifiée sur tous les évènements sauf e.
- Les nouvelles extensions de β' sont celles ajoutées par e. Elles se situent causalement après e, donc leur date de tir maximum est plus élevée que celle de e. Or, puisque β est temporellement complet, les évènements de β vérifient déjà la contrainte 6.11 pour la date de tir maximum de e, donc ils la vérifieront *a fortiori* pour les extensions de e.
- Pour le nouvel évènement e, puisque sa date de tir maximum était la plus faible, la date de tir choisie est nécessairement plus faible que la date de tir maximum des autres extensions de β ; pour les nouvelles extensions, puisqu'elles se situent causalement après e, la contrainte 6.11 est trivialement vérifiée.

Nous pouvons maintenant construire n'importe quel processus de branchement temporel en partant d'un préfixe et en ajoutant les extensions nécessaires.

Proposition 6.6 *Étant donné un processus de branchement temporel $\langle \beta, \nu, \theta \rangle$ temporellement complet et une extension β' de β telle que $\beta \subset \beta'$, il existe une temporisation θ' qui est une extension de θ, telle que $\langle \beta', \nu, \theta' \rangle$ soit un processus de branchement temporel temporellement complet.*

Preuve 6.6 (Proposition 6.6) *On démontre cette proposition par induction en ajoutant successivement à β les évènements de β' non présents dans β. Ainsi, à chaque pas on ajoute une extension à β en choisissant celle dont la date de tir maximum est la plus faible. On s'assure ainsi que le processus de branchement temporel est toujours temporellement complet. On peut alors appliquer récursivement la proposition 6.5, ce qui construit β' avec une fonction de temporisation θ' valide qui étend θ.*

Nous remarquons cependant que par ce processus on ne peut pas obtenir toutes les temporisations valides de β'. En effet il peut exister des temporisations θ' de β' qui lorsqu'elles

sont restreintes à β ne constituent pas une temporisation valide de β.

6.2.2.3 Configurations d'un processus de branchement temporel

Étant données une valuation des paramètres ν et une temporisation θ, le processus de branchement temporel $\langle B, E, F, l, \nu, \theta\rangle$ définit une configuration $E_{<\infty} = \{e \in E \mid \theta(e) \neq \infty\}$.

Proposition 6.7 *$E_{<\infty}$ est une configuration de $\beta = \langle B, E, F, l\rangle$, c'est-à-dire que :*
- *pour tout $e \in E_{<\infty}$, $\forall e' \in E,\ e' < e \Rightarrow e' \in E_{<\infty}$,*
- *pour tous $e, e' \in E_{<\infty}$, $\neg(e \# e')$.*

Preuve 6.7 (Proposition 6.7)

1. *$\forall e \in E_{<\infty}$, $\forall e' < e$, d'après le lemme 6.4 on a $\theta(e') \leq \theta(e)$. Comme $e \in E_{<\infty}$, $\theta(e) \neq \infty$ et donc nécessairement $\theta(e') \neq \infty$, ce qui entraîne par définition que $e' \in E_{<\infty}$.*
2. *$\forall e, e' \in E_{<\infty}$, supposons $e \# e'$. Alors, d'après le lemme 6.2 il existe deux chemins $se_1 \dots e$ et $se'_1 \dots e'$ tels que e_1 conf e'_1. D'après le premier point, $E_{<\infty}$ est causalement clos et donc on a $e_1, e'_1 \in E_{<\infty}$. Donc $\theta(e_1) \neq \infty$ et $\theta(e'_1) \neq \infty$. Or $\theta(e_1)$ doit vérifier les inéquations 6.7 ou 6.8 ce qui entraîne dans tous les cas que $\theta(e'_1) = \infty$ d'où la contradiction.*

Pour $e \in E_{<\infty}$ on montre également avec le lemme 6.4 que $\mathsf{Earlier}\,(e) \subseteq E_{<\infty}$ est une configuration.

Dans un processus de branchement d'un réseau de Petri un évènement ne dépend que de son passé causal décrit par la configuration locale $[e]$. À l'opposé, dans un processus temporel les contraintes de validité de la fonction de temporisation font intervenir tous les évènements avant e dans le processus. Dans le dépliage de TPN défini par Chatain et Jard, un évènement dépend des conditions qu'il consomme, mais également des conditions lues. Son passé causal temporel est donc plus grand que sa configuration locale car il contient également le passé causal de ces conditions lues.

Il en est de même avec les évènements d'un processus de branchement temporel. Les contraintes données dans la définition 6.7 font intervenir les conditions consommées, mais également les évènements libérateurs et les conditions consommées par un évènement en conflit direct. Pour un évènement d'un processus de branchement temporel on définit donc la notion de *configuration locale temporelle* qui étend celle de configuration locale en prenant aussi en compte ces dépendances sur la date de tir de l'évènement. Ainsi, la configuration locale temporelle notée $\lceil e \rceil$ d'un évènement e, regroupe récursivement en partant de e tous les évènements dont les dates de tirs apparaissent dans les contraintes de la définition 6.7.

Pour un évènement e d'un processus de branchement temporel, la *configuration locale temporelle* de e est :

$$\lceil e \rceil = [e] \bigcup_{\{e' \mid e' \ \mathsf{conf}\ e \vee e' \ \mathsf{lib}\ e\}} \lceil e' \rceil$$

Pour un évènement $e \in E_{<\infty}$, sa configuration locale ne contient que des évènements dont la date de tir est finie : $[e] \subseteq E_{<\infty}$. Ce n'est par contre pas le cas de la configuration locale temporelle de e qui peut très bien contenir des évènements dont la date de tir est infinie. Cela permet de représenter d'autres dépendances causales dues à l'introduction du temps. Ces dépendances ne sont pas inscrites dans la structure du processus de branchement correspondant au dépliage, mais elles sont liées à une temporisation choisie.

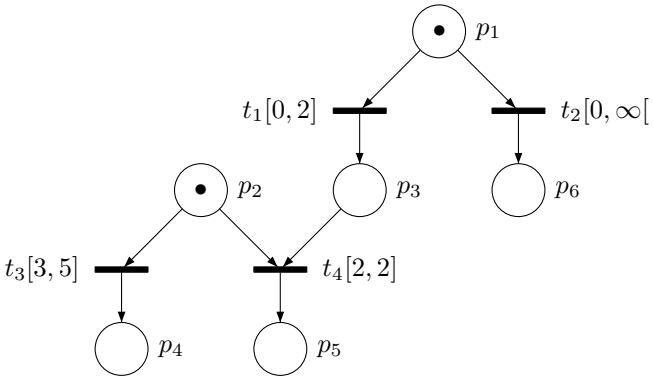


FIG. 6.8: Enchaînement de conflits

Exemple 6.8 *Ainsi sur le TPN de la figure 6.8, le tir de t_3 dépend d'après l'inéquation 6.7 de la date de production de la condition correspondant à p_3, c'est-à-dire de la date de tir de t_1. Or selon la temporisation considérée, cette date de tir aura soit une valeur finie comprise entre $[0,2]$, soit une valeur infinie dans le cas où c'est la transition t_2 qui est franchie.*

Cela introduit de nouvelles relations de causalité. Si dans le conflit entre t_1 et t_2 c'est la transition t_1 qui est franchie, le tir de t_3 est alors possible entre les dates 3 et 4 ; après ce délai le tir de t_4 est forcé. Pour pouvoir tirer t_3 à la date 5, il est donc nécessaire que t_2 soit tirée, c'est-à-dire que la date de tir de t_1 soit infinie.

Ces dépendances sont donc liées à la temporisation choisie. Pour pouvoir calculer l'ensemble des dates de tir possibles, on sera amené à distinguer pour la transition t_3, le domaine temporel obtenu lorsque t_1 est tirée, et celui obtenu lorsque t_2 est tirée. Cela revient en quelque sorte à dupliquer temporellement l'évènement correspondant au tir de la transition.

6.2.3 Relations entre processus de branchement temporels et processus temporels

Pour prouver la correction et la complétude du dépliage effectué par un processus de branchement temporel, nous montrons premièrement que tous les processus temporels que l'on peut extraire de ce dépliage sont valides, et deuxièmement que tous les processus temporels valides peuvent être représentés dans un processus de branchement temporel. Pour prouver la correction il faut évidemment que le processus de branchement temporel soit temporellement complet.

Lemme 6.8 *Dans un processus de branchement temporel $\langle \beta, \nu, \theta \rangle$, avec $\beta = \langle B, E, F, l \rangle$, soit $e \in E$ tel que $\theta(e) \neq \infty$:*

$$\forall e' \in E,\ {}^\bullet e' \cap C_e \neq \emptyset \ \Rightarrow\ \theta(e) \leq \theta(e')$$

On rappelle que $C_e = \mathsf{Cut}\,(\mathsf{Earlier}\,(e))$.

Preuve 6.8 (Lemme 6.8) *Par l'absurde, si $\theta(e') < \theta(e)$ alors $e' \in \mathsf{Earlier}\,(e)$. On considère alors la suite d'évènements causals dans $\mathsf{Earlier}\,(e)$, $e_1 < e_2 < \cdots < e_n$, telle que $e_1 = e'$ (et donc $\forall 1 \leq i \leq n,\ \theta(e_i) < \theta(e)$). Puisqu'on suppose que le temps diverge cette suite est nécessairement finie. Il existe donc une valeur de n telle que $e_n^\bullet \subseteq C_e$. Soit $b_1 \in {}^\bullet e' \cap C_e$ et $b_n \in e_n^\bullet$. $b_1, b_n \in C_e$ et on a crée un chemin de b_1 à b_n. C'est absurde car C_e est une coupure.*

Théorème 6.9 (Correction) *Soit $\langle \beta, \nu, \theta \rangle$, avec $\beta = \langle B, E, F, l \rangle$, un processus de branchement temporel temporellement complet d'un réseau de Petri temporel paramétré $\mathcal{N} = \langle P, T, Par, {}^\bullet(.), (.)^\bullet, M_0, J_s, D_p \rangle$. On considère $E_{<\infty} = \{e \in E \mid \theta(e) \neq \infty\}$. $\theta_{<\infty}$ est la restriction de θ à $E_{<\infty}$.*
$\langle E_{<\infty}, \theta_{<\infty} \rangle$ est alors un processus temporel valide.

Preuve 6.9 (Théorème 6.9) *On a déjà montré avec la proposition 6.7 que $E_{<\infty}$ est une configuration. Il faut maintenant montrer que $\theta_{<\infty}$ est une fonction temporisation valide pour ce processus.*

L'inéquation 6.5 montre directement la contrainte sur la date minimum : $\forall e \in E_{<\infty}$, $\theta(e) \geq \mathsf{TOE}({}^\bullet e, l(e)) + \mathsf{eft}(l(e))$.

Pour la date maximum, $\forall e \in E_{<\infty}$, $\forall t \in \mathsf{enabled}(l(C_e))$ on montre que : $\theta(e) \leq \mathsf{TOE}(C_e, t) + \mathsf{lft}(t)$.
On note $B' = \{b \in C_e \mid l(b) \in {}^\bullet t\}$ l'ensemble des conditions qui sensibilisent t.

- *Si $t = l(e)$ alors d'après l'équation 6.6 on a : $\theta(e) \leq \mathsf{TOE}({}^\bullet e, l(e)) + \mathsf{lft}((l(e))$. Puisque ${}^\bullet e \subseteq C_e$ et puisque qu'on suppose le réseau sauf, les conditions sensibilisant t dans C_e sont exactement ${}^\bullet e$, donc $\mathsf{TOE}({}^\bullet e, l(e)) = \mathsf{TOE}(C_e, l(e))$, ce qui prouve la formule.*
- *Sinon, si $\exists e' \in E$ t.q. $l(e') = t$ et ${}^\bullet e' = B'$, nécessairement d'après le lemme 6.8 : $\theta(e) \leq \theta(e')$.*
  1. *Si $\theta(e') \neq \infty$ alors d'après l'inéquation 6.6 : $\theta(e') \leq \mathsf{TOE}({}^\bullet e', l(e')) + \mathsf{lft}(l(e')) = \mathsf{TOE}(C_e, l(e')) + \mathsf{lft}(l(e'))$. On déduit le résultat cherché.*
  2. *Sinon, $\theta(e') = \infty$ et donc une des disjonctions 6.9 ou 6.10 est vérifiée. Mais puisque ${}^\bullet e' = B' \subseteq C_e$, les dates de productions de ses préconditions sont nécessairement finies, et donc on ne peut pas trouver de solution à l'équation 6.9. Ceci entraîne nécessairement qu'une des disjonctions 6.10 est vérifiée, c'est-à-dire : $\exists e'' \in E$ t.q. e'' conf e' et $\theta(e'') \neq \infty$. D'après le lemme 6.3, il existe un autre évènement $e^{(3)}$ tel que $e^{(3)}$ conf e' et $e^{(3)}$ vérifie l'inéquation 6.7. On en déduit que $e^{(3)} \in E_{<\infty}$. Par définition du conflit, $e^{(3)}$ possède alors au moins une précondition commune avec e' ; donc ${}^\bullet e^{(3)} \cap C_e \neq \emptyset$. On peut alors appliquer à nouveau le lemme 6.8, ce qui implique que $\theta(e) \leq \theta(e^{(3)})$, et puisque $e^{(3)}$ vérifie l'inéquation 6.7 on obtient : $\theta(e^{(3)}) \leq \mathsf{TOE}({}^\bullet e', l(e')) + \mathsf{lft}(l(e'))$, d'où le résultat.*
- *Dans le cas où aucun évènement dans β ne correspond au tir de t, alors t est une extension possible e' de β puisqu'il existe des conditions $B' \subseteq C_e$ qui sensibilisent t. Dans ce cas l'inéquation 6.11 assurant que β est un processus temporellement complet permet de déduire que : $\theta(e) \leq \mathsf{TOE}(B', t) + \mathsf{lft}(t) = \mathsf{TOE}(C_e, t) + \mathsf{lft}(t)$.*

Théorème 6.10 (Complétude) *Étant donné un réseau de Petri temporel paramétré $\mathcal{N} = \langle P, T, Par, {}^\bullet(.), (.)^\bullet, M_0, J_s, D_p \rangle$ et $\nu \in D_p$ une valuation des paramètres, soit $\langle E, \theta \rangle$ un processus temporel valide du réseau de Petri temporel $[\![\mathcal{N}]\!]_\nu$. E est une configuration d'un processus de branchement $\beta = \langle E^\bullet, E, F, l \rangle$.*
Alors il existe un processus de branchement temporel $\langle B', E', F', l', \nu, \theta' \rangle$, tel que $\forall e \in E$, $\exists e' \in E'$, t.q. $l(e) = l'(e')$ et $\theta(e) = \theta'(e')$. De plus, ce processus de branchement temporel est temporellement complet.

Preuve 6.10 (Théorème 6.10) *Pour prouver le théorème on construit un processus de branchement temporel $\Sigma' = \langle B', E', F', l', \nu, \theta' \rangle$, initialement égal à β, auquel on ajoute les*

évènements en conflit avec des évènements de E, en leur associant comme date de tir ∞ :

$$\forall e' \in PE(\beta),\ (\exists e \in E,\ e \ \mathsf{conf}\ e') \ \Rightarrow\ (e' \in E' \wedge \theta'(e') = \infty)$$

On rajoute aussi dans B' les conditions produites par e' et on étend les fonctions F' et l' en conséquence. La proposition 6.1 assure que Σ respecte la structure de processus de branchement.

On vérifie maintenant que ce processus de branchement temporel vérifie les contraintes sur les dates de tir des évènements. $\forall e \in E'$: si $\theta(e) = \infty$ alors par construction $\exists e' \in E'$ t.q. e conf e' et $\theta'(e') \neq \infty$; $\theta(e)$ vérifie alors l'équation 6.10. Sinon $e \in E$ et $\theta'(e) = \theta(e) \neq \infty$:

1. *Puisque θ est une temporisation valide pour E, la date minimum (inéquation 6.1) vérifie : $\theta(e) \geq \mathsf{TOE}\,({}^\bullet e, l(e)) + \mathsf{eft}\,(l(e))$ et donc $\theta'(e)$ vérifie l'inéquation 6.5.*
2. *De même pour la date maximum (inéquation 6.2) : $l(e) \in \mathsf{enabled}\,(l(C_e)) \Rightarrow \theta(e) \leq \mathsf{TOE}\,({}^\bullet e, l(e)) + \mathsf{lft}\,(l(e))$ et donc $\theta'(e)$ vérifie l'inéquation 6.6.*
3. *Ensuite, $\forall e' \in E'$, t.q. e' conf e, par construction $\theta'(e') = \infty$, et :*
   - *si ${}^\bullet e' \subset C_e$, alors $l(e') \in \mathsf{enabled}\,(l(C_e))$, et puisque θ est valide $\theta(e) \leq \mathsf{TOE}\,(C_e, l(e')) + \mathsf{lft}\,(l(e'))$ et l'inéquation 6.7 est alors vérifiée ;*
   - *sinon, si $\exists b \in {}^\bullet e'$ tel que $\theta'({}^\bullet b) \geq \theta'(e)$, c'est-à-dire que b est produite après C_e : on obtient immédiatement $\theta'(e) \leq \max_{b \in {}^\bullet e'}(\theta'({}^\bullet b)) + \mathsf{lft}\,(l(e'))$.*
   - *sinon, cela signifie qu'il existe une condition $b \in {}^\bullet e'$ qui se situe avant C_e : donc cette condition est consommée par un évènement $e'' \in b^\bullet$ qui appartient à $\mathsf{Earlier}\,(e)$. On a donc $e''\#e'$ ce qui entraîne d'après le lemme 6.2 qu'il existe $e''_1 \leq e''$,$e'_1 \leq e'$ tels que e''_1 conf e'_1. Cependant tous les évènements précédents e'' et e' doivent être dans E qui est une configuration, et qui donc n'admet pas de conflit. Le seul évènement qui n'est pas dans E est e', donc nécessairement $e'_1 = e'$. On a donc e''_1 conf e' et e' conf e. De plus nécessairement e''_1 co e : puisqu'ils appartiennent tous deux à E ils ne sont pas en conflit ; et si $e''_1 < e$ alors les conditions de e' ne sont plus concurrentes. On en conclut que $(e''_1, e', e) \in$ lib . Or $e''_1 \in \mathsf{Earlier}\,(e)$, donc $\theta'(e''_1) < \theta(e)$. Cela permet à e de vérifier l'inéquation 6.8.*

   On a donc construit un processus de branchement temporel Σ' valide, qui contient par construction l'ensemble des évènements de E.
4. *On montre également que Σ' est temporellement complet. Pour toute extension $e \in PE(\Sigma')$: si e consomme des conditions dont la date de production est infinie, alors la condition 6.11 est trivialement vérifiée. Sinon si ${}^\bullet e \not\subseteq \mathsf{Cut}\,(E)$, alors $\exists e' \in E$ tel que ${}^\bullet e' \neq \emptyset$; le lemme 6.2 entraîne comme précédemment qu'il existe $e'' \leq e'$ tel que e conf e'' ; mais alors cela signifie que e' a déjà été rajouté dans Σ' par le processus de construction. Sinon ${}^\bullet e' \subseteq \mathsf{Cut}\,(E)$ et donc e' est aussi une extension de E. Dans ce cas, puisque θ est valide, E est un processus temporellement complet, et donc la condition 6.3 est vérifiée pour l'extension e de E, ce qui implique trivialement la condition 6.11 pour Σ'.*

6.2.4 Marquages accessibles

Dans le cas non temporel, nous avons vu que les marquages accessibles d'un réseau de Petri correspondent aux coupures du dépliage. Les marquages accessibles d'un réseau de

Petri temporel consituent un sous-ensemble de ces marquages. Il faut donc non seulement que les conditions forment une coupure, mais de plus il faut vérifier que ces conditions peuvent coexister à une date commune.

Proposition 6.11 *Étant donné un réseau de Petri temporel paramétré $\mathcal{N}$ et un processus de branchement temporel complet $\langle \beta, \nu, \theta \rangle$, avec $\beta = \langle B, E, F, l \rangle$, soit B' une coupure de β telle que :*

- $\forall b \in B',\ \theta({}^\bullet b) \neq \infty$ *et,*
- $\forall e \in E,\ (\exists b \in B',\ b < e) \Rightarrow (\max\{\theta({}^\bullet b) \mid b \in c\} \leq \theta(e))$,

Alors $l(B')$ est un marquage accessible de $\mathcal{N}_\nu$.

Preuve 6.11 *Soit $E' = \{e \in \beta \mid \exists b \in B',\ e < b\}$: E' est l'union des configurations locales des producteurs des conditions B'. E' est une configuration et $B' = \mathsf{Cut}\,(E')$. On a également $E_{<\infty} = \{e \in E \mid \theta(e) \neq \infty\}$. Nécessairement $E' \subseteq E_{<\infty}$, car si toutes les dates de production des conditions B' sont finies, cela signifie (récursivement d'après les inéquations 6.5) que les dates de tirs des évènements causals sont finies. D'après le théorème 6.9 cette configuration permet de construire un processus temporel $\langle E_{<\infty}, \theta_{<\infty} \rangle$ valide pour $\mathcal{N}_\nu$.*

On considère alors un entrelacement ρ des évènements de $E_{<\infty}$. Soit n l'indice maximum dans ρ des évènements de E'. On suppose que $\nexists k < n$, t.q. $\theta(e_k) = \theta(e_n)$ et $e_k \notin E'$ (en considérant un entrelacement quelconque cette condition pourra être vérifiée entre changeant l'ordre des évènements dont la date de tir est égale $\theta(e_n)$). Nous montrons alors par l'absurde que $\forall e_k \in E_{<\infty}$ si $k < n$ alors $e_k \in E'$. Si $e_k \notin E'$, alors $\exists e'_k \leq e_k$ et tel que ${}^\bullet e_{k'} \subseteq B'$: en effet puisque B' est une coupure elle décrit un état maximal, et donc e_k possède nécessairement dans son passé un évènement qui ne consomme que des conditions de B'. Comme $k' < n$, alors par définition de ρ $\theta(e_{k'}) \leq \theta(e_n)$. La condition rajoutée sur le choix de l'entrelacement ρ entraîne même que $\theta(e_{k'}) < \theta(e_n)$, ce qui est une absurdité car par hypothèse sur B', puisque ${}^\bullet e_{k'} \subseteq B'$, $(\max\{\theta({}^\bullet b) \mid b \in B'\} \leq \theta(e_{k'}))$, et donc en particulier $\theta(e_n) \leq \theta(e_{k'})$.

Donc, on peut trouver un entrelacement ρ tel qu'il existe un entier k tel que $E' = E_k$ (où $E_k = \{e_i \in E_{<\infty} \mid i \leq k\}$). Et alors d'après le théorème 15 de [Aura 97] $M_k = l(\mathsf{Cut}\,(E_k)) = l(E')$ est un marquage accessible de $\mathcal{N}_\nu$ par la séquence de transition $FS(\rho)$.

6.2.5 Dépliage symbolique d'un réseau de Petri temporel paramétré

Nous considérons maintenant l'ensemble des temporisations valides pour un processus de branchement $\beta = \langle E, B, F, l \rangle$ d'un réseau de Petri temporel paramétré ; c'est-à-dire l'ensemble des couples de valuations $(\nu|\theta)$ pour les paramètres et les dates de tir des évènements de β, tels que $\langle \beta, \nu, \theta \rangle$ est un processus de branchement temporel. Cet ensemble de valuations forme un domaine $D \subseteq (\mathbb{R}^+ \cup \{\infty\})^{Par \times E}$. Ce domaine est défini symboliquement par l'ensemble des contraintes de la définition 6.7.

Nous définissons le dépliage complet d'un réseau de Petri temporel paramétré. Il contient l'ensemble des évènements possibles et l'ensemble des temporisations valides.

Définition 6.8 (Dépliage d'un réseau de Petri temporel paramétré) *Le dépliage $\mathcal{U}(\mathcal{N}) = \langle \beta_\mathcal{U}, D_\mathcal{U} \rangle$ d'un réseau de Petri temporel paramétré $\mathcal{N}$ est défini par le plus grand processus de branchement étendu constructible $\beta_\mathcal{U} = \langle E_\mathcal{U}, B_\mathcal{U}, F_\mathcal{U}, l_\mathcal{U} \rangle$ (au sens de la relation de préfixe), associé à l'ensemble des couples de valuations $(\nu|\theta)$ pour les paramètres et les dates de tir des évènements de $\beta_\mathcal{U}$, tels que $\langle \beta_U, \nu, \theta \rangle$ est un processus de branchement temporel. L'ensemble de ces valuations compose le domaine $D_\mathcal{U} \subseteq (\mathbb{R}^+ \cup \{\infty\})^{Par \times E_\mathcal{U}}$.*

$\beta_{\mathcal{U}}$ peut également être défini comme un processus de branchement étendu ne possédant pas d'extensions : tel que $PE(\beta_{\mathcal{U}}) = \emptyset$. Le domaine $D_{\mathcal{U}}$ est quant à lui décrit par les contraintes définissant le processus temporel $\langle \beta_U, \nu, \theta \rangle$. On remarque que pour tout $(\nu|\theta) \in D_{\mathcal{U}}$, $\langle \beta_U, \nu, \theta \rangle$ est nécessairement temporellement complet.

Le théorème de correction s'applique en particulier au dépliage du réseau de Petri pour chaque couple de valeurs $(\nu|\theta)$. En considérant un processus temporel, le théorème de complétude démontre que l'on peut construire un processus de branchement temporel complet avec les mêmes évènements. En appliquant la proposition 6.6 au dépliage on démontre alors que ce processus de branchement temporel est inclus dans le dépliage du réseau de Petri.

Ce dépliage, *a priori* infini, contient autant d'évènements que le dépliage du réseau de Petri non temporel sous-jacent. Par contre certains évènements ne peuvent avoir pour valeur temporelle que l'infini, ce qui signifie qu'ils ne sont pas possibles. En pratique, il ne sera pas nécessaire de poursuivre le dépliage après ces évènements. Notre dépliage temporel est donc nécessairement plus petit que celui du réseau de Petri sous-jacent. Cela constitue clairement une amélioration par rapport au dépliage défini par Chatain et Jard, pour lequel à cause de la duplication de nombreux évènements aucune borne sur la taille du dépliage n'est connue. Il faut toutefois noter que certaines des duplications introduites dans le dépliage de Chatain et Jard se retrouvent traduites dans notre approche par une duplication des valeurs temporelles associées aux évènements, comme c'est le cas pour le réseau de la figure 6.8 lorsqu'un enchaînement de conflits entraîne de multiples causes d'indéterminisme pour un évènement.

Préfixe complet Un préfixe du dépliage d'un réseau de Petri paramétré $\mathcal{N}$ sera dit *complet* s'il contient l'ensemble des marquages accessibles du réseau de Petri. Un préfixe $\langle \beta, D \rangle$ est un préfixe complet du dépliage $\mathcal{U}(\mathcal{N})$ si :

- β est un préfixe de $\beta_{\mathcal{U}}$;
- pour toute valuation ν des paramètres, pour tout marquage M accessible dans $[\![\mathcal{N}]\!]_\nu$, il existe une coupure B' de β telle que $l(B') = M$ et :
  - $\forall b \in B',\ \theta(^\bullet b) \neq \infty$ et,
  - $\forall e \in E,\ (\exists b \in B',\ b < e) \Rightarrow (\max\{\theta(^\bullet b) \mid b \in c\} \leq \theta(e))$,

Nous ne possédons pas de condition nécessaire et suffisante pour déterminer un préfixe fini complet du dépliage d'un réseau de Petri temporel paramétré. Nous pouvons utiliser des conditions suffisantes : si un évènement reproduit l'ensemble des conditions du marquage initial il peut être déclaré « cut-off » et le dépliage peut être arrêté après l'évènement, tout comme dans l'algorithme de [McMillan 95] pour les réseaux de Petri non temporel : en effet, tous les marquages atteints après cet évènement auront été atteints avant. On peut alors déterminer un préfixe complet du dépliage s'il se termine uniquement par des évènements « cut-off » ou ne possédant pas d'extensions.

Exemple 6.9 *Nous reprenons le TPN de la figure 6.1. Un préfixe du dépliage de ce modèle est donné sur la figure 6.9. Les contraintes symboliques sur les dates de tir des évènements sont écrites à côté de chaque évènement. Ce préfixe est de plus complet : en effet les évènements e_1, e_2, e_3 ne possèdent pas d'extensions possibles, et l'évènement e_4 réinitialise le marquage initial du réseau ; on est donc certain de posséder tous les marquages accessibles.*

Nous pouvons comparer ce dépliage avec celui obtenu par la méthode de Chatain et Jard qui est présenté sur la figure 6.4. Par notre méthode, le dépliage obtenu est structurellement identique au dépliage non temporel. Ainsi le tir de la transition t_3 n'est pas dupliqué et est

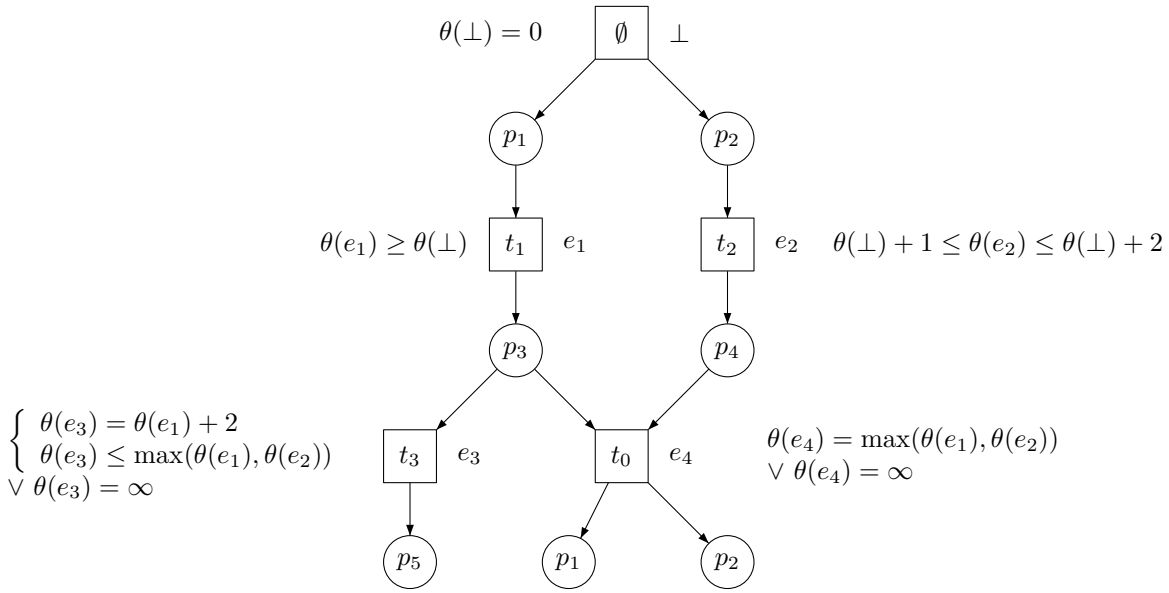


FIG. 6.9: Préfixe du dépliage symbolique du TPN 6.1

décrit uniquement par l'évènement e_3. Au contraire, sur le dépliage de la figure 6.4 les deux évènements e_3 et e_5 correspondent tous les deux au tir de t_3, en distinguant le cas où p_2 est marquée de celui où p_4 est marquée. Dans notre méthode, nous profitons du fait que la transition t_2 est nécessairement tirée, ce qui active le conflit entre e_3 et e_4. L'analyse des contraintes temporelles qui déterminent le tir de t_3 peut alors être effectuée directement au niveau du conflit direct, en ayant de connaissance de toutes les conditions intervenant dans ce conflit (à savoir p_3 et p_4).

6.3 Conclusion

Nous avons présenté dans chapitre notre contribution au développement des méthodes de dépliage des réseaux de Petri temporels. Dans ce domaine une méthode de dépliage a déjà été proposée par Chatain et Jard. Notre méthode essaye d'apporter certaines améliorations en adoptant une approche différente. Elle est basée sur la structure des processus de branchement, et pour déterminer les contraintes temporelles sur les évènements nous analysons les situations de conflits entre les évènements.

Le premier intérêt de la méthode est de définir un dépliage plus compact qui possède la même taille que le dépliage du réseau de Petri sous-jacent (on se contentera même d'un préfixe de ce dépliage ne contenant que les évènements qui sont possibles à une date finie). On évite de cette manière de devoir dupliquer des évènements, ce qui limite les risques d'explosion combinatoire. Mais en contrepartie, la résolution des contraintes temporelles pour déterminer les dates de tirs possibles d'un évènement devient plus complexe. On doit d'une part séparer les valeurs finies des valeurs infinies, ce qui revient d'une certaine façon à dupliquer un évènement (ces duplications sont toutefois strictement limitées à l'indéterminisme dans le passé de l'évènement). D'autre part, nous ne possédons plus de condition nécessaire et suffisante pour

calculer les dates de tirs. Nous adoptons donc une approche optimiste, mais il est nécessaire de recalculer par la suite les dates de tir de certains évènements.

Enfin, une des contributions est d'appliquer cette méthode de dépliage à des réseaux paramétrés. Cela est facilité par la manipulation symbolique des domaines temporels. En résolvant un problème avec la méthode de dépliage il sera alors possible de synthétiser des contraintes sur les paramètres.

Chapitre 7

Application de la méthode de dépliage à un problème de supervision

Résumé : *Dans ce chapitre nous présentons l'implémentation dans* Romeo *de la méthode de dépliage des réseaux de Petri temporels paramétrés. Les principaux algorithmes concernent d'une part les aspects structurels (analyse de la concurrence, détermination des conflits), et d'autre part les aspects temporels afin de déterminer les dates de tirs possibles pour les évènements.*
Nous appliquons notre méthode au problème de la supervision sur un modèle paramétré, dans lequel on cherche à déterminer les exécutions expliquant un ensemble d'observations, et à synthétiser les contraintes sur les paramètres permettant ces exécutions. Nous illustrons ce problème par l'étude d'un modèle paramétré du protocole du bit alterné.

Sommaire

7.1 Algorithmes de dépliage

Nous décrivons dans le semi-algorithme 7.1 la procédure de construction du dépliage d'un PTPN. Puisque ce dépliage est *a priori* infini, ce semi-algorithme ne termine pas en général. Les hypothèses supposées sur le PTPN sont les suivantes :

- il est sauf ;
- il n'est pas zénon.

Le principe de l'algorithme est celui décrit par la proposition 6.1 : on construit un processus de branchement en ajoutant au fur et à mesure de nouveaux évènements correspondant aux extensions du processus de branchement.

- Pour chaque nouvel évènement e ajouté, grâce à la procédure `DéterminerConflits` on détermine les nouvelles relations de conflit et de libération qui sont créées.
- La procédure `DéterminerDatesDeTir` calcule les dates de tirs possibles du nouvel évènement, et modifie celles des évènements déjà présents dont la date de tir est influencée par e.
- On calcule enfin les extensions du nouveau processus de branchement créé : les extensions possibles grâce à e sont calculées par la fonction `Extensions` et sont ajoutées à la liste des extensions PE.

Algorithme 7.1 : Dépliage temporel paramétré

Données : N : réseaux de Petri temporel paramétré
Résultat : $\mathcal{U}(\mathcal{N}) = \langle E, B, F, l, D \rangle$
Globales : $E, B, F, l, D, \mathsf{conf}, \mathsf{lib}$

```
1  début
2      E ← ⊥
3      B ← ⊥•
4      conf ← ∅
5      lib ← ∅
6      PE ← Extensions(⊥)
7      Tant que ∃e ∈ PE Faire
8          PE ← PE\{e}
9          U ← U ∪ {e}
10         B ← B ∪ e•
11         DéterminerConflits(e, )
12         DéterminerDatesDeTir(e)
13         PE ← PE ∪ Extensions(e)
14     Fin Tant que
15 fin
```

Nous détaillons dans la suite les différentes fonctions utilisées : une première catégorie de fonctions dépend uniquement des aspects structurels, c'est-à-dire non temporels. Les aspects temporels sont eux principalement étudiés lors de la détermination des dates de tirs des évènements.

7.1.1 Algorithmes non temporels

Dans cette section sont présentés les algorithmes utilisés pour construire un processus de branchement correspondant au dépliage du réseau de Petri temporel paramétré. Ces algorithmes sont indépendants des contraintes temporelles sur les dates de tir des évènements. On construit de cette manière la structure utilisée pour déplier le PTPN.

7.1.1.1 Calcul des extensions d'un processus de branchement

L'algorithme 7.2 calcule les extensions possibles après un évènement e. Pour le processus de branchement β tel que $e \in \beta$, on calcule ainsi l'ensemble $\{e' \in PE(\beta) \mid \exists b \in e^\bullet \cap {}^\bullet e'\}$. Si $\beta = \beta' \cup \{e\}$, on détermine grâce à cette fonction les extensions du processus de branchement β, en ajoutant les extensions possibles après le nouvel évènement e à la liste des extensions de β' qui est elle déjà calculée.

Le principe de l'algorithme est de déterminer toutes les extensions possibles qui consomment un ensemble de conditions concurrentes, dont au moins une produite par e. Pour cela, on essaye pour chaque transition consommant un jeton dans une des places de $l(e^\bullet)$ de trouver un ensemble de conditions B' permettant le tir de cette transition. Pour éviter d'ajouter plusieurs fois une même extension, on pourra fixer un ordre total entre les conditions. Dans ce cas, pour chaque condition b dans $e^\bullet$, on détermine les transitions tirables qui n'utilisent que des conditions inférieures à b. L'algorithme utilise une fonction `SontConcurrents` qui détermine si un ensemble de conditions (ou d'évènements, ou plus généralement de nœuds dans un processus de branchement) sont en concurrence (*i.e.* en relation par `co` par paire).

Algorithme 7.2 : Extensions

Données : e : nouvel évènement
Résultat : pe : extensions possibles après e

```
1  début
2      Pour Chaque b ∈ e• Faire
3          Pour Chaque t ∈ l(b)• Faire
4              Pour Chaque B' ⊆ B Faire
5                  Si l(B') = •t ET SontConcurrents(B') Alors
6                      Soit e tel que •e = B' et l(e) = t.
7                      pe ← pe ∪ {e}
8                  Fin Si
9              Fin Pour
10         Fin Pour
11     Fin Pour
12     Retourner pe
13 fin
```

7.1.1.2 Détermination des relations de conflits

L'algorithme 7.3 de détermination des conflits calcule les nouveaux conflits engendrés par l'ajout d'un évènement e au dépliage. Pour déterminer les relations de conflit direct, on doit

Algorithme 7.3 : DéterminerConflits

Données : e : nouvel évènement
Résultat : Mise à jour des relations de conflits

```
1  début
2      Pour Chaque b ∈ •e Faire
3          Pour Chaque e' ∈ b• Faire
4              Si SontConcurrents(•e ∪ •e') Alors
5                  conf ← (e, e')
6                  Pour Chaque e'' ≠ e t.q. e'' conf e' Faire
7                      Si SontConcurrents({e, e''}) Alors
8                          lib ← (e, e', e'')
9                      Fin Si
10                 Fin Pour
11                 Pour Chaque e'' ≠ e' t.q. e'' conf e Faire
12                     Si SontConcurrents({e', e''}) Alors
13                         lib ← (e', e'')
14                     Fin Si
15                 Fin Pour
16             Fin Si
17         Fin Pour
18     Fin Pour
19 fin
```

rechercher les évènements partageant des conditions d'entrée avec e, et tels que leurs préconditions sont concurrentes. Les nouveaux conflits directs sont ajoutés à la liste conf définissant la relation de conflit. Chaque nouveau conflit direct peut alors engendrer de nouvelles relations de libération, qui sont ajoutées à la liste lib.

7.1.1.3 Détermination de la concurrence

Dans les procédures structurelles présentées précédemment le point critique est l'analyse de la concurrence entre les nœuds du dépliage. On utilise pour cela la procédure récursive de parcours du dépliage présentée dans [Chatain 06a].

La procédure `SontConcurrents` doit déterminer si un ensemble X de nœuds du dépliage (conditions ou évènements) sont concurrents. Elle se contente d'appeler, pour chacun des nœuds $x \in X$, la procédure récursive `Traverse` définie par l'algorithme 7.4. Le principe est de parcourir le dépliage et d'annoter les nœuds déjà visités en les stockant dans la liste *visited* ; si au cours du parcours on repasse par une condition déjà annotée alors les nœuds ne sont pas concurrents.

7.1.2 Détermination des dates de tir des évènements

Nous donnons maintenant les principes utilisés pour déterminer les dates de tirs des évènements ce qui permet de spécifier la procédure `DéterminerDatesDeTir`. Pour connaître les valeurs valides des dates de tirs des évènements, il est nécessaire de résoudre le système de

Algorithme 7.4 : Traverse

Données : x : nœuds (condition ou évènement), *visited* : liste des nœuds déjà visités
Résultat : VRAI si aucune condition n'est déjà visitée dans le passé causal de x, FAUX sinon

```
1  début
2      Si x ∈ E Alors
3          Si x ∉ visited Alors
4              visited ← visited ∪ {x}
5              Pour Chaque b ∈ •x Faire
6                  Si NON Traverse(b) Alors Retourner FAUX
7              Fin Pour
8          Fin Si
9          Retourner VRAI
10     Sinon
11         Si x ∈ visited Alors Retourner FAUX
12         Sinon Si NON Traverse(•x) Alors Retourner FAUX
13         visited ← visited ∪ {x}
14         Retourner VRAI
15     Fin Si
16 fin
```

contraintes définissant un processus de branchement temporel. Ce système de contraintes définit un domaine $D \subseteq (\mathbb{R}^+ \cup \{\infty\})^{Par \times E}$ contenant l'ensemble des dates de tir possibles pour les évènements E du dépliage.

Afin de représenter ce domaine, il est possible de séparer les valeurs finies et les valeurs infinies des dates de tir. À chaque partition P de l'ensemble E des évènements en deux sous-ensembles $E_{<\infty}$ et $E_{=\infty}$, on associe un domaine $D_P \subseteq \mathbb{R}^{+E_{<\infty}}$, qui ne contient donc plus que des dates finies. Ce domaine est défini par les contraintes de la définition 6.7 dans lesquelles nous avons éliminé les inéquations utilisant des dates de tir infinies. Les conditions restantes concernent alors la structure du processus de branchement. Elles font intervenir des opérateurs max qui peuvent être éliminés et remplacés par des disjonctions ; les quantificateurs universels sont remplacés par des conjonctions ; les quantificateurs existentiels par des disjonctions. Au final, chaque domaine D_P peut donc être représenté par une union de polyèdres. Le domaine D peut alors être représenté par l'union des domaines D_P qui sont chacun associés à un ensemble de conditions définies par $E_{=\infty}$.

Nous constatons par ailleurs que la date de tir d'un évènement e ne dépend que des dates de tir des évènements appartenant à sa configuration locale temporelle $\lceil e \rceil$. On peut donc déterminer les dates de tir possibles de e en résolvant uniquement les contraintes posées sur les évènements de $\lceil e \rceil$. La procédure `DéterminerDatesDeTir` calcule de cette manière, à chaque ajout d'un nouvel évènement e au dépliage, les dates de tir possibles pour e.

Cependant, l'ajout d'un évènement e au dépliage peut également modifier les dates de tir des évènements qui étaient déjà présents dans le dépliage, en l'occurrence celles des évènements e' tels que $e \in \lceil e' \rceil$. En effet, lors de l'ajout de l'évènement e, de nouvelles contraintes sont ajoutées aux évènements e'' tels que e'' conf e ou e'' lib e, ce qui modifie également

les évènements suivants. La procédure `DéterminerDatesDeTir` calcule alors les modifications nécessaires.

Enfin, en calculant les dates de tir d'un évènement, on détermine si l'évènement est *possible*, c'est-à-dire qu'il admet des valeurs finies pour sa date de tir. Certains évènements ne peuvent en effet posséder qu'une date de tir infinie. À partir de ces évènements *impossibles*, on déduit immédiatement que tous les évènements suivants (causalement) sont impossibles. En pratique, nous n'ajouterons pas au dépliage les extensions consommant des conditions produites par un évènement impossible. En ajoutant cette condition, le semi-algorithme 7.1 de dépliage ne calcule qu'un préfixe du dépliage. Ce préfixe est toutefois complet (les évènements impossibles qui ne sont pas ajoutés ne permettent pas d'accéder à un marquage du PTPN), et donc de taille moins importante que le dépliage du réseau de Petri sous-jacent.

7.2 Étude de cas d'un problème supervision

7.2.1 Problème de la supervision

Les dépliages peuvent être utilisés pour vérifier des propriétés d'accessibilité dans les réseaux de Petri, par exemple à l'aide de l'algorithme de Mc Millan qui calcule un préfixe fini complet du dépliage. Dans les réseaux de Petri temporels la méthode de Chatain et Jard permet également de déterminer un préfixe fini complet, mais le critère permettant de tronquer le dépliage est beaucoup plus compliqué à appliquer. Dans les réseaux de Petri temporels paramétrés les problèmes d'accessibilité sont indécidables et nous ne possédons pas de critère permettant de déterminer un préfixe fini complet du dépliage.

Mais d'autres applications pour les dépliages ont été étudiées. C'est le cas notamment du problème de la *supervision basée sur les modèles*. Il s'agit s'effectuer un diagnostic d'un système à évènements discrets, en déterminant quels enchaînements d'évènements dans le modèle permettent d'expliquer des observations issues de capteurs. Ces observations sont constituées par la liste des évènements observés sur le système. En général, ces évènements correspondent à certaines transitions du modèle, alors que les autres transitions sont des transitions internes non observables (on parle alors d'observation partielle dans la théorie du contrôle supervisé [Cassandras 08]). Les dépliages sont très intéressants dans ce contexte, car ils permettent de déterminer les relations de causalités entre les évènements ce qui ne figure pas dans les observations. En considérant des dépliages temporels paramétrés on détermine de plus les contraintes temporelles entre les dates de tir d'évènements, ainsi que des contraintes sur les paramètres. Les problèmes du contrôle supervisé et du diagnostic à l'aide de réseaux de Petri ont été étudiés dans de nombreux papiers (par exemple [Ushio 98] et [Giua 97]), dans lesquels l'analyse est pour la plupart basée sur le graphe d'état, c'est-à-dire avec une approche séquentielle utilisant des entrelacements. L'utilisation des dépliages est plus récente. Dans [Fabre 05] les auteurs utilisent des réseaux de Petri ordinaires saufs, et se focalisent sur le diagnostic distribué. Dans [Chatain 05] les auteurs utilisent les dépliages de réseaux de Petri temporels saufs pour réaliser une supervision temporelle.

Pour réaliser la supervision d'un système à l'aide des dépliages, on construit « à la volée » un préfixe du dépliage en se guidant avec les observations. On exclut ainsi du dépliage les évènements incompatibles avec les observations. On détermine par ce biais des listes partiellement ordonnées d'évènements du dépliage, telles que l'ensemble de leurs évènements expliquent entièrement les observations : elles sont alors appelées des *explications*. Puisque les

observations sont en nombre finies, la décidabilité du problème devient plus facile à obtenir. Il reste à régler le cas possible des boucles de transitions non observables : nous choisissons de résoudre ce problème en fixant un nombre maximum d'évènements non observables. Une alternative peut être de considérer des modèles avec probabilités afin de ne déterminer que les explications les plus probables, en supposant que la répétition des boucles d'évènements non observables fait décroître la probabilité d'une trajectoire [Benveniste 03].

7.2.1.1 Modélisation du problème de la supervision

Le modèle d'un système est construit à l'aide d'un réseau de Petri temporel paramétré. L'ensemble des évènements observables dans le système est décrit par un alphabet fini d'actions Σ. Afin de faire correspondre les observations à des transitions du modèle, nous ajoutons à chaque transition du réseau une étiquette correspondant à une action. Les transitions non observables seront étiquetées par une action spéciale notée τ également ajoutée à Σ.

Définition 7.1 (Réseau de Petri temporel paramétré étiqueté) *Un* réseau de Petri temporel paramétré étiqueté *est un 10-uplet* $\mathcal{N} = \langle P, T, Par, \Sigma, \lambda, {}^\bullet(.), (.)^\bullet, M_0, J_s, D_p \rangle$ *tel que :*

- $\langle P, T, Par, {}^\bullet(.), (.)^\bullet, M_0, J_s, D_p \rangle$ *est un PTPN,*
- $\Sigma = \{\tau, a_1, \ldots, a_n\}$ *est un alphabet fini d'*actions*, dans lequel* τ *est une* action muette*,*
- $\lambda \in \Sigma^T$ *est une fonction qui associe à chaque transition une* étiquette *dans* Σ*.*

Les observations issues des capteurs du système définissent une séquence finie $\sigma \in \Sigma^*$. Nous intégrons dans cette séquence le nombre maximum d'évènements non observables pris en compte en ajoutant des actions τ. Afin de guider la construction du dépliage par ces observations, nous utilisons la fonction de Parikh de la séquence σ, définie par :

$$\varpi : \Sigma^* \rightarrow \mathbb{N}^{|\Sigma|}$$

Cette fonction associe à une séquence d'observations σ un vecteur de Parikh $\varpi(\sigma)$, dans lequel on compte le nombre d'occurrences dans σ de chaque action de Σ.

Pour chaque évènement e d'un processus de branchement $\langle B, E, F, l \rangle$, on calcule également un vecteur de Parikh $\zeta(e) \in \mathbb{N}^{|\Sigma|}$, qui compte cette fois-ci le nombre d'occurrences de chaque action dans le passé causal de e. Pour calculer $\zeta(e)$ nous définissons pour chaque action $a \in \Sigma$ le vecteur de Parikh χ_a tel que chacune de ses composantes est égale à (0), sauf celle correspondant à l'action a qui est égale à (1). On peut alors calculer $\zeta(e)$ par :

$$\zeta(e) = \sum_{e' \in [e]} \chi_{\lambda(l(e))}$$

En comparant les vecteurs de Parikh des évènements avec le vecteur de Parikh des observations, nous nous assurerons qu'un évènement est compatible avec les observations, et dans le cas contraire nous pourrons l'exclure du dépliage.

7.2.1.2 Calcul du superviseur

Le *superviseur* $\mathcal{E}(\mathcal{N}, \sigma)$ du système modélisé par un PTPN étiqueté $\mathcal{N}$, pour une séquence d'observations σ, est un préfixe $\langle \beta, D \rangle$ du dépliage $\mathcal{U}(\mathcal{N})$. Il est calculé par l'algorithme 7.1 de dépliage, mais en excluant les évènements qui ne font pas partie d'une explication. On

ne conserve donc que les évènements e tels que $\zeta(e) \leq \varpi(\sigma)$. Pour cela lors de l'ajout des extensions issues d'un évènement e à la liste PE des extensions, on élimine celles dont le vecteur de Parikh dépasse $\varpi(\sigma)$.

De plus, pour que le superviseur soit complet, c'est-à-dire qu'il contient l'ensemble des explications valides pour la séquence d'observations, nous lui ajoutons toutes ses extensions qui sont en conflit avec des évènements déjà présents. Le nouveau superviseur calculé est alors $\langle \beta', D' \rangle$, tel que $\beta \subseteq \beta'$, et $\forall e \in PE(\beta)$, $e \in \beta'$ ssi $\exists e' \in \beta$ t.q. e' conf e. Ces extensions ajoutées à β' ne seront évidemment compatibles avec aucune explication, mais en les ajoutant on augmente le nombre des temporisations de $\langle \beta', D' \rangle$ qui seront temporellement complètes, suffisamment pour que $\langle \beta', D' \rangle$ contienne toutes les explications compatibles.

7.2.1.3 Extraction des explications

Pour un PTPN étiqueté $\mathcal{N}$, une séquence d'observations σ et une valuation ν des paramètres, une *explication* est une exécution possible dans $[\![\mathcal{N}]\!]_\nu$ expliquant l'ensemble des actions observables de σ. Elle peut être décrite par un processus temporel valide $\langle E, \theta \rangle$ de $[\![\mathcal{N}]\!]_\nu$, tel que la somme des vecteurs de Parikh de ses évènements soit égale au vecteur de Parikh des observations pour toutes les composantes correspondant aux actions observables, *i.e.* :

$$\forall a \in \Sigma,\ a \neq \tau,\ (\sum_{e \in E} \chi_{\lambda(l(t))})(a) = (\varpi(\sigma))(a) \tag{7.1}$$

La composante de l'action non observable τ doit elle être inférieure à $(\varpi(\sigma))(\tau)$.

En calculant le superviseur $\mathcal{E}(\mathcal{N}, \sigma) = \langle \beta, D \rangle$ nous pouvons extraire l'ensemble des explications de σ dans $\mathcal{N}$. Pour une valuation $(\nu|\theta) \in D$ telle que $\langle \beta, \nu, \theta \rangle$ est temporellement complet, $\langle E_{<\infty}, \theta_{<\infty} \rangle$ définit d'après le théorème de correction 6.9 un processus temporel valide. On peut alors extraire de ce processus les sous-ensembles d'évènements $E' \subseteq E_{<\infty}$ causalement clos, expliquant l'ensemble des observations, c'est-à-dire vérifiant l'équation 7.1. Si de plus E' est un processus temporellement complet pour la temporisation $\theta_{<\infty}$ alors E' décrit une explication de σ dans $\mathcal{N}$.

Inversement, pour une valuation ν des paramètres, chaque explication possible est décrite par un processus temporel valide $\langle E, \theta \rangle$ de $[\![\mathcal{N}]\!]_\nu$. D'après le théorème de complétude 6.10 on forme en rajoutant à ce processus les extensions qui sont en conflit avec des évènements de E, un processus de branchement temporel $\langle \beta', \theta' \rangle$ temporellement complet. Par construction, β' est un préfixe du superviseur β, et d'après la proposition 6.5 il existe une temporisation θ valide pour β qui est une extension de θ'. Nous prouvons ainsi que chaque explication possible est incluse dans le superviseur $\mathcal{E}(\mathcal{N}, \sigma) = \langle \beta, D \rangle$.

Si l'on considère maintenant les dates de tirs de façon symbolique, nous pouvons directement extraire du superviseur $\mathcal{E}(\mathcal{N}, \sigma) = \langle \beta, D \rangle$ les ensembles d'évènements E' causalement clos et expliquant l'ensemble des observations, *i.e.* vérifiant l'équation 7.1. On ajoute au domaine D les contraintes 6.4 pour ne garder que les valuations temporellement complètes. On calcule alors le domaine $D' \subset D$ tel que $\forall (\nu|\theta) \in D'$, $\forall e \in E'$, $\theta(e) \neq \infty$. On rajoute alors au domaine D' la contrainte 6.5 de complétude d'un processus temporel. Le domaine D' finalement obtenu décrit l'ensemble des dates de tirs des évènements de E' tel que E' constitue une explication au problème de supervision. De plus, en projetant D' sur l'ensemble des para-

mètres du modèle on obtient une contrainte sur les paramètres qui conditionne l'accessibilité de cette explication.

7.2.2 Étude de cas

Nous analysons dans cette étude de cas un problème de supervision sur un modèle paramétré du protocole du bit alterné. Nous avons modélisé ce système dans Romeo par un PTPN, dans lequel nous utilisons des paramètres temporels pour représenter les temps de transmission et les délais de retransmission. Étant donnée une séquence d'observations sur ce système, le but est de déterminer l'ensemble des explications pour ces observations, et de synthétiser les contraintes sur les paramètres conditionnant ces explications. Cet exemple a déjà été traité dans [Grabiec 09], mais en modélisant le système par un réseau d'automates temporisés. La méthode utilisée par les auteurs consiste à construire dans un premier temps un préfixe du dépliage non temporel du modèle. Ensuite, pour chaque explication possible extraite du dépliage, les auteurs déterminent si des temporisations sont valides pour l'ensemble des évènements. Finalement, les contraintes sur les paramètres peuvent être déterminées à partir des contraintes temporelles calculées. Nous allons au contraire appliquer la méthode présentée dans ce chapitre en calculant directement un préfixe du dépliage temporel paramétré.

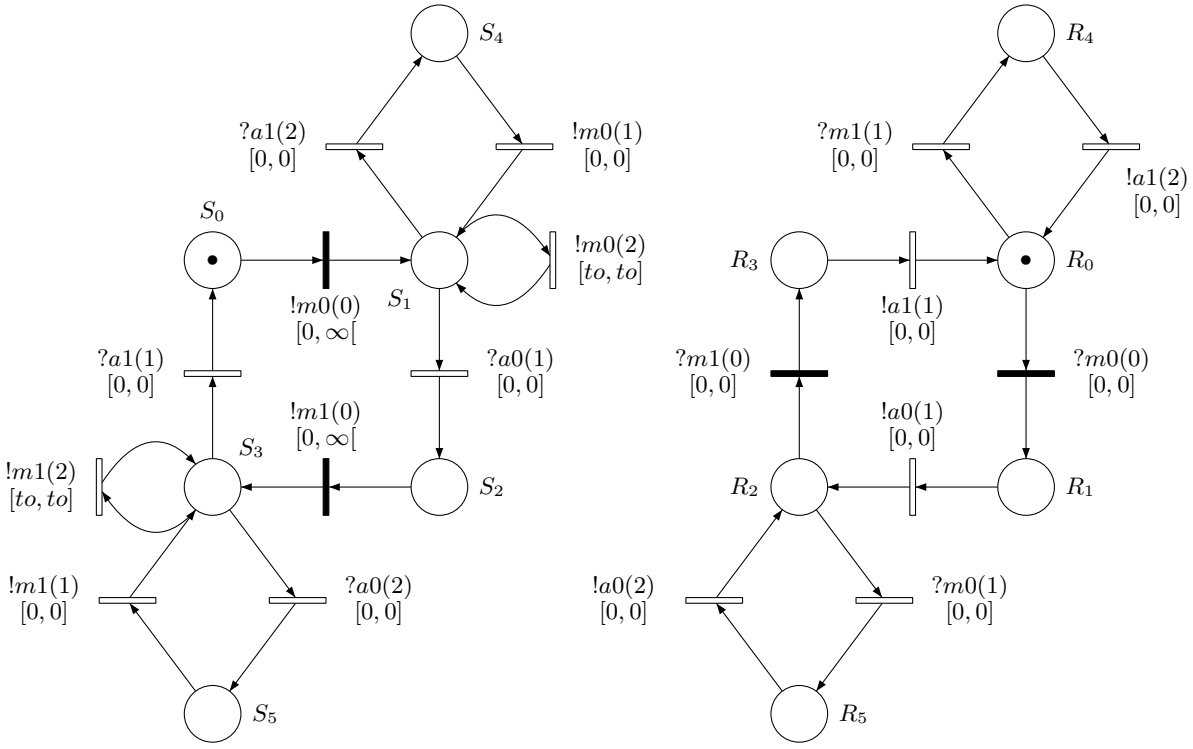
Le modèle PTPN du système est constitué d'un émetteur $\mathcal{N}_S$ présenté sur la figure 7.1a. Pour $i \in [0..2]$, les transitions $!m0(i)$ émettent un message avec un bit (0), alors que celles notées $!m1(i)$ émettent un message avec un bit (1). Les transitions de réception des accusés sont $?a0(1)$, $?a0(2)$, $?a1(1)$ et $?a1(2)$. Le délai de retransmission des messages est fixé par un paramètre to. Sur ce module, les seules transitions observables sont $!m0(0)$ et $!m1(0)$. Elles correspondent à la première émission d'un message (avec un bit (0) ou (1)). Elles sont dessinées par un rectangle plein sur la figure du PTPN. On considère qu'elles sont étiquetées par le nom de la transition. Toutes les autres transitions sont non observables ; on considère donc qu'elles sont étiquetées par l'action τ (non notée sur la figure).

Le récepteur $\mathcal{N}_R$ est quant à lui présenté sur la figure 7.1b. De manière analogue, les transitions de réception d'un message sont $?m0(i)$ et $?m1(i)$ (pour $i \in [0..1]$), et celles d'émission d'un accusé sont $!a0(i)$ et $!a1(i)$ (pour $i \in [1..2]$). Dans ce module, les seules transitions observables correspondent à la première réception d'un message, c'est à dire $?m0(0)$ et $?m1(0)$.

Enfin, ces deux modules sont reliés par deux canaux de communication, un dédié aux messages, l'autre aux accusés. Chacun de ces canaux correspond au modèle PTPN de la figure 7.2. Ces canaux de communication sont modélisés par une file d'attente, dont la taille est limitée à deux messages. Les transitions t_i ($i \in [1..6]$) sont des transitions internes qui assurent l'acheminement des messages. Les transitions $!x0$ et $!x1$ signalent l'arrivée d'un message dans le canal, alors que les transitions $?x0$ et $?x1$ délivrent le message. La perte d'un message est possible par la transition $l(0)$ ou $l(1)$.

Le canal de transmission des messages est décrit par le modèle $\mathcal{N}_T(m)$. Dans ce modèle les transitions $!x0$, $!x1$, $?x0$ et $?x1$ sont notée respectivement $!m0$, $!m1$, $?m0$ et $?m1$. Les paramètres temporels x et X sont remplacés par deux paramètres m et M, qui sont respectivement le temps minimum et le temps maximum nécessaire à la transmission d'un message.

Similairement, le canal de transmission des accusés de réception est décrit par le modèle $\mathcal{N}_T(a)$. Les transitions $!x0$, $!x1$, $?x0$ et $?x1$ y sont notées $!a0$, $!a1$, $?a0$ et $?a1$, et les paramètres



(a) Émetteur $\mathcal{N}_S$ (b) Récepteur $\mathcal{N}_R$

FIG. 7.1: Émetteur et récepteur du protocole du bit alterné

temporels x et X sont remplacés par a et A. a est le temps minimum de transmission d'un accusé, et A le temps maximum.

Le modèle PTPN complet du protocole de bit alterné, noté $\mathcal{N}$, est obtenu en effectuant la composition synchrone des modèles $\mathcal{N}_S$, $\mathcal{N}_R$, $\mathcal{N}_T(m)$ et $\mathcal{N}_T(a)$. Pour cela, les transitions $!m0$, $!m1$, $?m0$ et $?m1$ de $\mathcal{N}_T(m)$ (resp. $!a0$, $!a1$, $?a0$ et $?a1$ de $\mathcal{N}_T(a)$) sont dupliquées pour être regroupées avec l'une des transitions correspondantes dans $\mathcal{N}_S$ et $\mathcal{N}_R$, à savoir $!m0(i)$, $!m1(i)$, $?m0(i)$ ou $?m1(i)$ (resp. $!a0(i)$, $!a1(i)$, $?a(i)$ ou $?a1(i)$). Les autres transitions et les places de $\mathcal{N}_T(m)$ (resp. $\mathcal{N}_T(a)$) sont préfixée par m (resp. a). Par construction, le modèle final est sauf. Il comporte 27 places, 33 transitions, dont seulement 4 sont observables ($!m0(0)$, $!m1(0)$, $?m0(0)$ et $?m1(0)$), et 5 paramètres temporels (to, m, M, a et A). On considère le domaine suivant pour les paramètres temporels :

$$D_p = \left\{ \begin{array}{l} 0 < a \leq A \\ 0 < m \leq M \\ 0 < to \end{array} \right\}$$

Résultats de la supervision du système

Nous considérons la séquence d'observations suivante :

$$\sigma = !m0(0), !m1(0), ?m0(0), ?m1(0), ?m0(0)$$

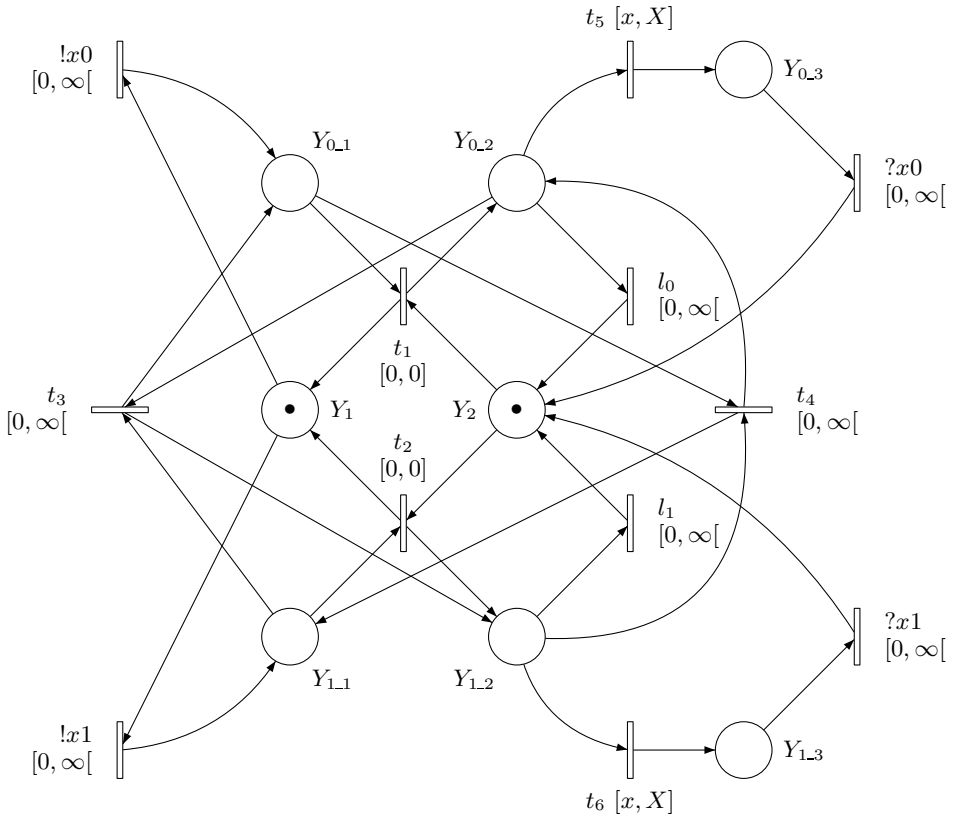


FIG. 7.2: Canal de transmission $\mathcal{N}_T(x)$ du protocole du bit alterné

En recherchant des explications à cette séquence, nous voulons déterminer s'il est possible d'envoyer un message (par $!m0(0)$) et de recevoir deux fois ce même message (par $?m0(0)$). Si cela est possible, nous aimerions savoir comment modifier le système afin d'éviter ces exécutions. Enfin, puisque le modèle intègre des transitions non observables, pour pouvoir calculer un résultat fini nous fixons un nombre maximum de 15 évènements non observables. Nous ajoutons donc à la séquence σ 15 occurrences de l'action muette τ.

Nous appliquons la méthode de supervision décrite précédemment, en commençant par calculer le superviseur $\mathcal{E}(\mathcal{N}, \sigma)$ par la méthode de dépliage temporel paramétré. Le préfixe du dépliage calculé contient 434 évènements possibles. Nous extrayons de ce superviseur 5 explications compatibles avec les observations. Chaque explication est conditionnée par une contrainte sur les paramètres temporels :

$$\text{Explication 1 : } \left\{ \begin{array}{l} m \leq A \\ m \leq to \\ m + a \leq 2to \\ m + a \leq to + M \\ to \leq M + A \end{array} \right\} \qquad \text{Explication 2 : } \left\{ \begin{array}{l} m \leq A \\ m \leq to \\ a \leq M \\ a \leq to \\ to \leq M + A \end{array} \right\}$$

$$\text{Explication 3 : } \left\{ \begin{array}{l} to \leq M + A \\ m + a \leq to + M \\ m + a \leq 2to \end{array} \right\}$$

$$\text{Explication 4 : } \left\{ \begin{array}{l} m \leq A \\ m \leq to \\ to \leq M + A \\ m + a \leq to + M \\ m + a \leq 2to \end{array} \right\}$$

$$\text{Explication 5 : } \left\{ \begin{array}{l} to \leq M \\ a \leq to \end{array} \right\} \vee \left\{ \begin{array}{l} to \leq M \\ m \leq to \\ m + a \leq 2to \end{array} \right\}$$

Nous pouvons déterminer à l'aide de ces contraintes la condition nécessaire suivante :

$$to \leq M + A$$

Afin de comparer nos résultats avec ceux obtenus dans [Grabiec 09], nous restreignons le domaine des paramètres de sorte que maintenant :

$$D_p = \left\{ \begin{array}{l} a = A \\ m = M \\ a = m \\ 0 < a \\ 0 < to \end{array} \right\}$$

Nous pouvons appliquer ces nouvelles conditions directement sur les résultats obtenus. Les quatre premières explications possèdent alors la même condition sur les paramètres :

$$a \leq to \leq 2a$$

La dernière explication possède une condition plus restreinte : $to = a$. Ce résultat est cohérent avec celui de [Grabiec 09] où les auteurs obtiennent également la condition nécessaire $to \leq 2a$. Cette condition peut être utilisée pour prévenir les exécutions correspondant à la séquence d'observations : en ajoutant la condition $to > 2a$ au domaine des paramètres on s'assure qu'un message ne peut pas être reçu deux fois. Nous notons que la condition $a \leq to$ est due quant à elle à la taille limitée des explications : dans cette limite de 15 évènements non observables il n'est pas possible de d'obtenir une exécution compatible avec les observations si le temps de retransmission est trop faible.

7.3 Conclusion

Ce chapitre conclut la partie présentant nos travaux sur le dépliages des réseaux de Petri temporels paramétrés. Nous y avons présenté les détails de l'implémentation de notre méthode de dépliage, et nous avons terminé en décrivant comment notre méthode pouvait être utilisée pour résoudre un problème de supervision. La contribution que nous apportons à la résolution de ce problème est de proposer une méthode permettant d'extraire les explications d'un ensemble d'observations, en calculant directement un dépliage temporel paramétré. Il est alors possible de déterminer des contraintes sur les paramètres qui conditionnent les explications calculées. Nous avons illustré cette approche sur un exemple du protocole du bit alterné, pour lequel le problème de la supervision a déjà été étudié dans la littérature. Nous retrouvons les résultats présentés dans la littérature, tout en mettant en valeur notre approche de résolution directe du problème.

Quatrième partie

Conclusion

Chapitre 8

Conclusion et perspectives

Bilan

Les travaux que nous avons réalisés ont porté sur l'étude de méthodes de vérification paramétrée. La motivation pour ces recherches a été de proposer des méthodes formelles pouvant être appliquées sur des systèmes dont les spécifications ne sont pas encore complètes. Cela nécessite d'introduire des paramètres dans les modèles utilisés afin de donner des degrés de liberté à la modélisation. Dans le but d'appliquer ces méthodes aux systèmes temps réel, nous nous sommes focalisé sur les paramètres temporels, qui sont en général parmi les plus complexes à définir. Nous avons donc défini le modèle des réseaux de Petri à chronomètres paramétrés dans lequel nous introduisons des paramètres temporels sur les transitions.

La première approche que nous avons suivie a été d'étendre les méthodes d'analyse classiquement utilisées dans les réseaux de Petri temporels. Nous avons donc défini une représentation de l'espace d'états de ces modèles paramétrés à l'aide d'un graphe des classes d'états paramétrées. En se basant sur l'exploration de ce graphe des classes, nous avons proposé des semi-algorithmes de model-checking paramétré avec lesquels nous pouvons vérifier des formules TCTL paramétrées. Nous montrons cependant que ce problème est indécidable. Pour une utilisation pratique il est donc nécessaire de limiter les degrés de libertés introduits par les paramètres : on peut pour cela limiter le nombre des paramètres et limiter leur domaine en ajoutant des contraintes supplémentaires.

L'étude de cas présentée illustre les possibilités offertes par cette approche. On s'aperçoit que la vérification paramétré peut servir à guider la conception du système : la vérification d'une propriété détermine un ensemble de valeurs admissibles pour les paramètres. Cela permet de restreindre le domaine des paramètres du modèle qui devient par ce processus plus précis. Nous pouvons alors vérifier de nouvelles propriétés.

La seconde approche que nous avons développée, cherche à utiliser le parallélisme présent dans les réseaux de Petri. Elle se base pour cela sur les méthodes de dépliages. Ces méthodes préservant la concurrence sont très intéressantes pour la vérification des systèmes distribués car elles permettent de limiter l'explosion combinatoire. Dans les modèles paramétrés le parallélisme est également un des facteurs qui restreint les possibilités d'utilisation des techniques de model-checking paramétré. Nous avons donc proposé une méthode de dépliage temporel paramétré dans laquelle nous limitons les entrelacements et les duplications des évènements.

Elle se base pour cela sur une analyse des conflits entre les évènements et sur une construction optimiste du dépliage, qui est *a priori* infini.

Nous proposons toutefois d'utiliser cette méthode pour résoudre un problème de supervision. Dans ce contexte, la construction du dépliage est guidée par des observations finies. Cela permet de construire un préfixe fini du dépliage. De cette manière, on explicite d'une part les causes logiques expliquant les observations, en construisant l'ordre partiel entre les évènements (les explications). D'autre, on détermine les causes temporelles, en synthétisant des contraintes sur les paramètres associées à ces explications.

Au vue de ces résultats nous pouvons suggérer le choix de la méthode à utiliser dans différentes situations. Le model-checking paramétré permet de vérifier des propriétés sans connaissance *a priori* sur le modèle. Il sera préféré pour vérifier des propriétés sur les marquages du réseau. Dans le cas où l'analyse est impossible, une exploration partielle de l'espace d'états paramétrés peut être utilisée pour tester certains scénarios. La technique de dépliage sera elle utilisée en conjonction avec des informations supplémentaires fixant des critères d'arrêt du dépliage, comme cela est le cas dans le problème de la supervision. Les dépliages sont alors intéressants pour déterminer les liens de causalités entre les évènements mais également pour déterminer des conditions temporelles paramétrées.

Perspectives

Parmi les extensions possibles à court terme des travaux de thèse, certaines améliorations peuvent être apportées au model-checking paramétré, afin d'augmenter l'efficacité de la vérification. Une des possibilités, si l'on se restreint à des réseaux de Petri temporels paramétrés, est d'utiliser des DBM paramétrées, tel que cela est fait pour les automates paramétrés [Hune 01], en remplacement des polyèdres dont la manipulation est coûteuse. Concernant la méthode de dépliage, nous n'avons pas inclus dans notre étude les réseaux à chronomètres. La méthode doit pouvoir être étendue au modèle des PITPN, en gérant en plus du calcul des dates de tir, le calcul du temps d'inhibition d'un évènement avant qu'il soit franchi.

Il serait également intéressant de se servir des dépliages temporels paramétrés pour effectuer du model-checking paramétré. Cependant, il faut pour cela disposer d'un critère efficace permettant de calculer un préfixe fini complet du dépliage. En adaptant le critère défini dans [Chatain 06b], il doit pouvoir s'appliquer à notre méthode. Mais pour appliquer ce critère il est nécessaire de comparer un état global atteint par le dépliage, ce qui est contraire à la démarche de conservation de la concurrence. Il serait intéressant de pouvoir disposer d'un critère plus local.

Dans des perspectives à moyen terme, nous nous sommes focalisés sur une étude des aspects temporels paramétrés. En poursuivant la motivation initiale de ces travaux, une piste de recherche est d'essayer de paramétrer d'autres aspects dans les modèles. Cela pourrait être utilisé pour définir plusieurs modes de fonctionnement d'un système sur un même modèle. Ainsi, dans les réseaux de Petri, il peut être intéressant de paramétrer la quantité de jetons présents dans une place.

Enfin, dans [Traonouez 07] nous envisagions d'utiliser les dépliages paramétrés dans une approche de vérification compositionnelle. La conservation de la concurrence devrait faciliter cette démarche qui nécessite d'être formalisée.

Bibliographie

[Abdulla 00] P. A. Abdulla, S. P. Iyer & A. Nylen. *Unfoldings of unbounded Petri nets*. In Proceedings of CAV, volume 1855 of *LNCS*, pages 495–507. Springer, 2000. 80

[Abdulla 01] Parosh Aziz Abdulla & Aletta Nylén. *Timed Petri Nets and BQOs*. In Proceedings of ICATPN, pages 53–70. Springer-Verlag, 2001. 21

[Alur 90] Rajeev Alur, Costas Courcoubetis & David L. Dill. *Model-Checking for Real-Time Systems*. In LICS, pages 414–425, 1990. 30, 42

[Alur 93] Rajeev Alur, Thomas A. Henzinger & Moshe Y. Vardi. *Parametric Real-time Reasoning*. In ACM Symposium on Theory of Computing, pages 592–601, 1993. 43, 46

[Alur 94] Rajeev Alur & David L. Dill. *A Theory of Timed Automata*. Theoretical Computer Science, vol. 126, pages 183–235, 1994. 5, 21, 30, 31

[Alur 95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. h. Ho, X. Nicollin, A. Olivero, J. Sifakis & S. Yovine. *The Algorithmic Analysis of Hybrid Systems*. Theoretical Computer Science, vol. 138, pages 3–34, 1995. 6, 25

[André 08] Étienne André, Thomas Chatain, Laurent Fribourg & Emmanuelle Encrenaz. *An Inverse Method for Parametric Timed Automata*. Electronic Notes in Theoretical Computer Science, vol. 223, pages 29–46, 2008. 44

[Annichini 00] Aurore Annichini, Eugene Asarin & Ahmed Bouajjani. *Symbolic Techniques for Parametric Reasoning about Counter and Clock Systems*. In Proceedings of CAV, pages 419–434. Springer-Verlag, 2000. 43

[Aura 97] Tuomas Aura & Johan Lilius. *Time Processes for Time Petri Nets*. In Proceedings of ICATPN, volume 1248 of *LNCS*, pages 136–155, 1997. 80, 85, 103

[Bagnara 98] Roberto Bagnara. *A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages*. Science of Computer Programming, vol. 30, no. 1-2, pages 119–155, 1998. 66

[Bagnara 08] R. Bagnara, P. M. Hill & E. Zaffanella. *The Parma Polyhedra Library : Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems*. Science of Computer Programming, vol. 72, no. 1–2, pages 3–21, 2008. 66

[Benveniste 03] A. Benveniste, E. Fabre, C. Jard & S. Haar. *Diagnosis of asynchronous discrete event systems, a net unfolding approach.* IEEE Transactions on Automatic Control, vol. 48, no. 5, pages 714–727, 2003. 114

[Bérard 05] Beatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime & Olivier (H.) Roux. *Comparison of the expressiveness of Timed Automata and Time Petri Nets.* In Proceedings of FORMATS, volume 3829 of *LNCS*, Uppsala, Sweden, 2005. Springer. 46

[Berthomieu 83] Bernard Berthomieu & Miguel Menasche. *An Enumerative Approach For Analyzing Time Petri Nets.* In Proceedings of IFIP, pages 41–46. Elsevier Science Publishers, 1983. 16, 25, 31, 34

[Berthomieu 91] B. Berthomieu & M. Diaz. *Modeling and verification of time dependent systems using time Petri nets.* IEEE Transactions on Software Engineering, vol. 17, no. 3, pages 259–273, 1991. 6, 21, 25, 31, 34, 47, 65

[Berthomieu 01] B. Berthomieu. *La méthode des Classes d'états pour l'Analyse des Réseaux Temporels - Mise en Oeuvre, Extension à la multi-sensibilisation.* In Modélisation des Systèmes Réactifs (MSR'01). Hermes, 2001. 21

[Berthomieu 03] Bernard Berthomieu & François Vernadat. *State Class Constructions for Branching Analysis of Time Petri Nets.* In Proceedings of TACAS, volume 2619 of *LNCS*, pages 442–457, Warsaw, Poland, April 2003. Springer Verlag. 32, 35

[Berthomieu 04] B. Berthomieu, P.-O. Ribet & F. Vernadat. *The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets.* International Journal of Production Research, vol. 42, no. 4, July 2004. Tool available at http ://www.laas.fr/tina/. 6

[Berthomieu 05] Bernard Berthomieu, Didier Lime, Olivier H. Roux & François Vernadat. *Problèmes d'Accessibilité et Espaces d'États Abstraits des Réseaux de Petri Temporels à Chronomètres.* In Modélisation des Systèmes Réactifs (MSR'05), Autrans, France, October 2005. 71, 73

[Berthomieu 07] Bernard Berthomieu, Didier Lime, Olivier H. Roux & François Vernadat. *Reachability Problems and Abstract State Spaces for Time Petri Nets with Stopwatches.* Discrete Event Dynamic Systems, vol. 17, no. 2, pages 133–158, 2007. 6, 25, 27, 34

[Best 88] E. Best & C.C. Fernandez. *Nonsequential processes : A Petri net view.* 1988. 79, 84

[Boucheneb 09] Hanifa Boucheneb, Guillaume Gardey & Olivier (H.) Roux. *TCTL model checking of Time Petri Nets.* Journal of Logic and Computation, 2009. 30, 35, 43, 53

[Boyer 01] Marc Boyer & Michel Diaz. *Multiple Enabledness of Transitions in Petri Nets with Time.* In Proceedings of PNPM, page 219, Washington, DC, USA, 2001. IEEE Computer Society. 21

[Boyer 08] Marc Boyer & Olivier (H.) Roux. *On the compared expressiveness of arc, place and transition time Petri Nets.* Fundamenta Informaticae, vol. 88, no. 3, pages 225–249, 2008. 21

[Brinksma 01] Ed Brinksma & Jan Tretmans. *Testing Transition Systems : An Annotated Bibliography*. In MOdelling and VErifying parallel Processes (MOVEP'01), volume 2067 of *LNCS*, pages 187–195. Springer, 2001. 5

[Bruyère 07] Véronique Bruyère & Jean-François Raskin. *Real-Time Model-Checking : Parameters everywhere*. CoRR, vol. abs/cs/0701138, 2007. 44

[Bucci 04a] Giacomo Bucci, Andrea Fedeli, Luigi Sassoli & Enrico Vicario. *Timed State Space Analysis of Real-Time Preemptive Systems*. IEEE Transactions on Software Engineering, vol. 30, no. 2, pages 97–111, 2004. 6, 25, 65, 71

[Bucci 04b] Giacomo Bucci, Luigi Sassoli & Enrico Vicario. *ORIS : A Tool For State-Space Analysis Of Real-Time Preemptive Systems*. In Proceedings of QEST, September 2004. 6

[Cassandras 08] C. G. Cassandras & S. Lafortune. Introduction to discrete event systems, second edition. Springer, 2008. 113

[Cassez 00] Franck Cassez & Kim Guldstrand Larsen. *The Impressive Power of Stopwatches*. In Proceedings of CONCUR, pages 138–152. Springer-Verlag, 2000. 6, 25

[Cassez 06a] F. Cassez, T. Chatain & C. Jard. *Symbolic unfoldings for networks of timed automata*. In Proceedings of ATVA, volume 4218 of *LNCS*, pages 307–321. Springer, 2006. 80

[Cassez 06b] Franck Cassez & Olivier (H.) Roux. *Structural Translation from Time Petri Nets to Timed Automata – Model-Checking Time Petri Nets via Timed Automata*. The journal of Systems and Software, vol. 79, no. 10, pages 1456–1468, 2006. 30, 42, 66

[Chatain 05] T. Chatain & C. Jard. *Time supervision of concurrent systems using symbolic unfoldings of time Petri nets*. In P. Pettersson & W. Yi, editeurs, Proceedings of FORMATS, volume 3829 of *LNCS*, pages 196–210. Springer, 2005. 113

[Chatain 06a] Thomas Chatain. *Symbolic Unfoldings of High-Level Petri Nets and Application to Supervision of Distributed Systems (Dépliages symboliques de réseaux de Petri de haut niveau et application à la supervision des systèmes répartis)*. PhD thesis, Université de Rennes 1, November 2006. 111

[Chatain 06b] Thomas Chatain & Claude Jard. *Complete Finite Prefixes of Symbolic Unfoldings of Safe Time Petri Nets*. In Proceedings of ICATPN, volume 4024 of *LNCS*, pages 125–145, june 2006. ix, 80, 87, 89, 94, 124

[de Frutos Escrig 00] David de Frutos Escrig, V. Valero Ruiz & O. Marroquin Alonso. *Decidability of Properties of Timed-Arc Petri Nets*. In Proceedings of ICATPN, pages 187–206. Springer-Verlag, 2000. 21

[Delfieu 07] David Delfieu, Médésu Sogbohossou, Louis-Marie Traonouez & Sébastien Revol. *Parameterized study of a Time Petri Net*. In Proceedings of CITSA, Orlando, Florida, USA, July 2007. 80, 89, 90

[Dill 89] 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. 16

[Emerson 82] E. Allen Emerson & Edmund M. Clarke. *Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons.* Science of Computer Programming, vol. 2, no. 3, pages 241–266, 1982. 29

[Emerson 86] E. Allen Emerson & Joseph Y. Halpern. *'Sometimes' and 'Not Never' Revisited : On Branching versus Linear Time Temporal Logic.* Journal of the Association for Computing Machinery, vol. 33, no. 1, pages 151–178, 1986. 29

[Engelfriet 91] Joost Engelfriet. *Branching processes of Petri nets.* Acta Informatica, vol. 28, no. 6, pages 575–591, 1991. 79, 82, 84

[Esparza 94] Javier Esparza. *Model checking using net unfoldings.* Science of Computer Programming, vol. 23, no. 2-3, pages 151–195, 1994. 79

[Esparza 96] Javier Esparza, Stefan Romer & Walter Vogler. *An Improvement of McMillan's Unfolding Algorithm.* In Proceedings of TACAS, pages 87–106, 1996. 79, 80, 85

[Esparza 08] J. Esparza & K. Heljanko. Unfoldings, a partial-order approach to model checking. Monographs in Theoretical Computer Science. Springer, 2008. 80

[Fabre 05] E. Fabre, A. Benveniste, S. Haar & C. Jard. *Distributed monitoring of concurrent and asynchronous systems.* Journal of Discrete Event Systems, special issue, pages 33–84, 5 2005. 113

[Gardey 03] Guillaume Gardey, Olivier (H.) Roux & Olivier (F.) Roux. *A zone-based method for Computing the State Space of a Time Petri Net.* In Proceedings of FORMATS, volume 2791 of *LNCS*, pages 246–259, Marseille, France, September 2003. Springer. 6, 32

[Gardey 05a] Guillaume Gardey. *Contribution à la vérification et au contrôle des systèmes temps réel – Application aux réseaux de Petri temporels et aux automates temporisés.* PhD thesis, Université de Nantes et École Centrale de Nantes, décembre 2005. 32, 35, 43, 53

[Gardey 05b] Guillaume Gardey, Didier Lime, Morgan Magnin & Olivier (H.) Roux. *Roméo : A Tool for Analyzing time Petri nets.* In Proceedings of CAV, volume 3576 of *LNCS*, Edinburgh, Scotland, UK, July 2005. Springer. 6, 65

[Gardey 06] Guillaume Gardey, Olivier (H.) Roux & Olivier (F.) Roux. *State Space Computation and Analysis of Time Petri Nets.* Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems, vol. 6, no. 3, pages 301–320, 2006. Copyright Cambridge Press. 43, 65, 66

[Girard 87] Jean Yves Girard. *Linear logic.* Theorical Computer Science, vol. 50, 1987. 80

[Girault 97] Francois Girault. *Formalisation en logique linéaire du fonctionnement des réseaux de Petri.* PhD thesis, Université Paul Sabatier, Toulouse, France, 1997. 80, 89

[Giua 97] A. Giua. *Petri net state estimators based on event observation.* In Proceedings of the IEEE ICDC, pages 4086–4091, 1997. 113

[Godefroid 96] Patrice Godefroid. Partial-order methods for the verification of concurrent systems : An approach to the state-explosion problem. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996. Foreword By-Wolper, Pierre. 79

[Grabiec 09] B. Grabiec & C. Jard. *Unfolding of networks of automata and their application in supervision.* In Proceedings of NOTERE, 2009. 116, 119

[Hadjidj 06] Rachid Hadjidj & Hanifa Boucheneb. *On-the-fly TCTL model checking for Time Petri Nets using state class graphs.* In Proceedings of ACSD, pages 111–122. IEEE Computer Society, 2006. 35, 43, 53, 66

[Hanisch 93] Hans-Michael Hanisch. *Analysis of Place/Transition Nets with Timed Arcs and its Application to Batch Process Control.* In Proceedings of ICATPN, pages 282–299. Springer-Verlag, 1993. 21

[Henzinger 94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis & Sergio Yovine. *Symbolic Model Checking for Real-Time Systems.* Information and Computation, vol. 111, pages 394–406, 1994. 6, 30

[Henzinger 97] Thomas A. Henzinger, Pei-Hsin Ho & Howard Wong-Toi. *HYTECH : A Model Checker for Hybrid Systems.* International Journal on Software Tools for Technology Transfer, vol. 1, no. 1–2, pages 110–122, 1997. 6, 44

[Hune 01] Thomas Hune, Judi Romijn, Mariëlle Stoelinga & Frits W. Vaandrager. *Linear Parametric Model Checking of Timed Automata.* In Proceedings of TACAS, volume 2031 of *LNCS*. Springer, 2001. 44, 47, 124

[Jones 77] N. D. Jones, L. H. Landweber & Y. E. Lien. *Complexity of Some Problems in Petri Nets.* Theoretical Computer Science 4, pages 277–299, 1977. 25, 27

[Khansa 96] W. Khansa, J.P. Denat & S. Collart-Dutilleul. *P-Time Petri Nets for Manufacturing Systems.* In Proceedings of the International Workshop on Discret Event Systems (Wodes'96), pages 94–102, Edinburgh (United Kingdom), August 1996. 21

[Khansa 97] W. Khansa. *Réseaux de Petri P-temporels, contribution à l'étude des systèmes à événements discrets.* PhD thesis, Université de Savoie, March 1997. 21

[Khomenko 03] V. Khomenko & M. Koutny. *Branching processes of high-level Petri nets.* In Proceedings of TACAS, volume 2619 of *LNCS*, pages 458–472. Springer, 2003. 80

[Komei 02] David A. Komei, David Avis, Komei Fukuda & Stefano Picozzi. *On Canonical Representations of Convex Polyhedra.* In Mathematical Software, World Scientific (2002) 351–360, pages 350–360, 2002. 16, 17

[Larsen 95] K. G. Larsen, P. Pettersson & W. Yi. *Model-Checking for Real-Time Systems.* In Fundamentals of Computation Theory, pages 62–88, 1995. 6, 14

[Larsen 97] K. G. Larsen, P. Pettersson & W. Yi. UPPAAL *in a Nutshell.* International Journal on Software Tools for Technology Transfer, vol. 1, no. 1–2, pages 134–152, Oct 1997. 6, 31, 41, 66

[Lime 03a] Didier Lime & Olivier H. Roux. *Expressiveness and analysis of scheduling extended time Petri nets.* In Proceedings of FET, Aveiro, Portugal, July 2003. Elsevier Science. Copyright Elsevier Science. 65

[Lime 03b] Didier Lime & Olivier H. Roux. *State class Timed Automaton of a Time Petri Net.* In Proceedings of PNPM, pages 124–133, Urbana, Illinois, USA, September 2003. IEEE Press. 66

[Lime 04a] Didier Lime. *Vérification d'applications temps réel à l'aide de réseaux de Petri temporels étendus.* PhD thesis, Université de nantes et École Centrale de Nantes, Nantes, France, December 2004. 27, 34

[Lime 04b] Didier Lime & Olivier (H.) Roux. *A translation based method for the timed analysis of scheduling extended time Petri nets.* In Proceedings of RTSS, pages 187–196, Lisbon, Portugal, December 2004. IEEE Press. 66

[Lime 06] Didier Lime & Olivier H. Roux. *Model checking of time Petri nets using the state class timed automaton.* Journal of Discrete Event Dynamic Systems (jDEDS), vol. 16, no. 2, pages 179–205, April 2006. 43

[Lime 09] Didier Lime, Olivier H. Roux, Charlotte Seidner & Louis-Marie Traonouez. *Romeo : A Parametric Model-Checker for Petri Nets with Stopwatches.* In Proceedings of TACAS, volume 5505 of *LNCS*, pages 54–57, York, United Kingdom, March 2009. Springer. 6, 63, 65

[Magnin 05] Morgan Magnin, Didier Lime & Olivier (H.) Roux. *An efficient method for computing exact state space of Petri nets with stopwatches.* In Third International Workshop on Software Model-Checking (SoftMC'05). Elsevier, July 2005. 65

[Magnin 07] Morgan Magnin. *Réseaux de Petri à chronomètres : temps dense et temps discret.* PhD thesis, École Centrale de Nantes, Nantes, France, December 2007. 27

[McMillan 95] K. L. McMillan. *A technique of state space search based on unfolding.* Formal Methods in System Design, vol. 6, no. 1, pages 45–65, 1995. 79, 85, 104

[Merlin 74] P.M. Merlin. *A study of the recoverability of computing systems.* PhD thesis, Department of Information and Computer Science, Univ. of California, Irvine, 1974. 6, 21

[Nielsen 80] M. Nielsen, G. Plotkin & G. Winskel. *Petri Nets, Event Structures and Domains.* Theorical Computer Science, vol. 13, no. 1, pages 85–108, January 1980. 79

[Penczek 01] Wojciech Penczek & Agata Pólrola. *Abstractions and Partial Order Reductions for Checking Branching Properties of Time Petri Nets.* In Proceedings of ICATPN, pages 323–342. Springer-Verlag, 2001. 79

[Penczek 04] Wojciech Penczek & Agata Pólrola. *Specification and Model Checking of Temporal Properties in Time Petri Nets and Timed Automata.* In Proceedings of ICATPN, pages 37–76, 2004. 43, 53

[Pezze 99] M. Pezze & M. Toung. *Time Petri nets : A primer introduction.* In Tutorial presented at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications, Zaragoza, Spain, September 1999. 21

[Pnueli 77] Amir Pnueli. *The temporal logic of programs.* In Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS'77), pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. 29

[Pradin-Chézalviel 99] Brigitte Pradin-Chézalviel, Robert Valette & Luis Allan Künzle. *Formalisation de scénarios, réseaux de Petri et logique linéaire.* In Journées FAC'99 : Formalisation des Activités Concurrentes, CERT-IRIT-LAAS, Toulouse, 1999. 80, 90

[Ramchandani 74] 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. 6, 21

[Rivière 03] Nicolas Rivière. *Modélisation et analyse temporelle par réseaux de Petri et logique linéaire.* Thèse de doctorat, Université Paul Sabatier, Toulouse, Novembre 2003. 80

[Roux 02] O.(H.) Roux & A-M. Déplanche. *A T-time Petri net extension for real time-task scheduling modeling.* European Journal of Automation (JESA), vol. 36, no. 7, pages 973–987, 2002. 6, 25, 65

[Roux 04] Olivier (H.) Roux & Didier Lime. *Time Petri Nets with Inhibitor Hyperarcs. Formal Semantics and State Space Computation.* In Proceedings of ICATPN, volume 3099 of *LNCS*, pages 371–390, Bologna, Italy, June 2004. Springer. 6, 25, 32, 47, 65

[Rushby 01] John M. Rushby. *Theorem Proving for Verification.* In MOdelling and VErifying parallel Processes (MOVEP'01), volume 2067 of *LNCS*, pages 39–57. Springer, 2001. 5

[Schrijver 86] Alexander Schrijver. Theory of linear and integer programming. John Wiley & Sons, Inc., New York, NY, USA, 1986. 16

[Toussaint 97] Joel Toussaint, Francoise Simonot-Lion & Jean-Pierre Thomesse. *Time Constraints Verification Methods Based on Time Petri Nets.* In FTDCS'97 : Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems, page 262, Washington, DC, USA, 1997. IEEE Computer Society. 28

[Traonouez 07] Louis-Marie Traonouez, David Delfieu & Olivier (H.) Roux. *Synthèse de contraintes de conception à partir de réseaux de Petri temporels paramétrés.* In Modélisation des Systèmes Réactifs (MSR'07), Lyon, France, October 2007. 124

[Traonouez 08] Louis-Marie Traonouez, Didier Lime & Olivier H. Roux. *Parametric Model-Checking of Time Petri Nets with Stopwatches Using the State-Class Graph*. In Proceedings of FORMATS, volume 5215 of *LNCS*, pages 280–294, Saint-Malo, France, September 2008. Springer. 39, 65

[Traonouez 09] Louis-Marie Traonouez, Didier Lime & Olivier (H.) Roux. *Parametric Model-Checking of Stopwatch Petri Nets*. Journal of Universal Computer Science, 2009. to appear. 39

[Tripakis 01] Stavros Tripakis & Sergio Yovine. *Analysis of Timed Systems Using Time-Abstracting Bisimulations*. Formal Methods in System Design, vol. 18, no. 1, pages 25–68, 2001. 43

[Ushio 98] T. Ushio, I. Onishi & K. Okuda. *Fault detection based on Petri net models with faulty behaviors*. In Proceedings of the IEEE Int. Conf. on SYstems, Man, and Cybernetics, pages 113–118, 1998. 113

[Vernadat 96] Francois Vernadat, Pierre Azema & Francois Michel. *Covering Step Graph*. In Proceedings of ICATPN, pages 516–535. Springer-Verlag, 1996. 79

[Virbitskaite 99] Irina Virbitskaite & E. Pokozy. *Parametric Behaviour Analysis for Time Petri Nets*. In Proceedings of PaCT, pages 134–140. Springer-Verlag, 1999. 44

[Wang 96] Farn Wang. *Parametric timing analysis for real-time systems*. Information and Computation, vol. 130, no. 2, pages 131–150, 1996. 44

[Yoneda 98] T. Yoneda & H. Ryuba. *CTL Model Checking of Time Petri Nets Using Geometric Regions*. IEICE Transactions on Information and Systems, vol. E81-D, no. 3, pages 297–396, 1998. 31, 35

[Yovine 97] Sergio Yovine. *Kronos : A Verification Tool for Real-Time Systems. (Kronos User's Manual Release 2.2)*. International Journal on Software Tools for Technology Transfer, vol. 1, pages 123–133, 1997. 6, 31

Annexe A

Preuves des semi-algorithmes de model-checking paramétré

Nous présentons dans cette annexe les preuves des semi-algorithmes de model-checking paramétré présentés au chapitre 4.

Pour deux états q_1, q_2, on utilisera dans les preuves les abréviations $\mathsf{time}(q_1) = \tau_1$ et $\mathsf{time}(q_2) = \tau_2$.

A.1 Semi-algorithme EU

Théorème A.1 *Soit $\mathcal{N}$ un PITPN et $\phi = \exists\varphi\ \mathcal{U}_J\ \psi$ une formule PTCTL telle que $J \in \mathcal{J}(Par)$. Étant donnée une classe d'états paramétrée étendue $C = (M, D)$ du graphe des classes d'états $\mathcal{G}_c(\mathcal{N})$, la correction et la complétude du semi-algorithme EU sont exprimées par :*

$$F^{\phi}(C) = F^{\phi}_{EU}(C)$$

Preuve A.1 (Théorème A.1 (correction)) *Nous considérons une valuation qui vérifie la formule logique calculée par l'algorithme : $\nu \in F^{\phi}_{EU}(C)$. Nous montrons que cette valuation vérifie bien le problème du model-checking paramétré, c'est-à-dire que : $\nu \in F^{\phi}(C) = \{\nu \in D_{|Par} \mid \exists q_1 \in [\![C]\!]_\nu,\ q_1 \models [\![\phi[J - \tau_1]]\!]_\nu\}$.*

De la formule $F^{\phi}_{EU}(C)$, on déduit directement que $\nu \in D_{|Par}$, ce qui implique que $[\![C]\!]_\nu \neq \emptyset$. On note aussi que $\nu \in \{\tau_{min}(C) \leq J^{\downarrow}\}$. Il existe ensuite 3 manières de vérifier la formule disjonctive :

1. *Si ν vérifie le premier terme de la disjonction (i.e.. $M \models \psi\ \wedge\ \{\tau_{max}(C) \geq {}^{\uparrow}J\}$).*
   On choisit dans ce cas q_1 t.q. ${}^{\uparrow}J(\nu) \leq \tau_1 \leq J(\nu)^{\downarrow}$, ce qui est possible puisque $\nu \in (\{\tau_{min}(C) \leq J^{\downarrow}\} \wedge \{\tau_{max}(C) \geq {}^{\uparrow}J\})$, et donc $[\tau_{min}(C)(\nu), \tau_{max}(C)(\nu)] \cap [{}^{\uparrow}J(\nu), J(\nu)^{\downarrow}] \neq \emptyset$. Soit alors une exécution $\rho \in \pi(q_1)$. Pour $r = 0$, nous avons $\rho^*(r) \models \psi$ et $r \in J(\nu) - \tau_1$. En conséquence $q_1 \models [\![\phi[J - \tau_1]]\!]_\nu$, ce qui prouve que $\nu \in F^{\phi}(C)$.
2. *Si ν vérifie le second terme de la disjonction, on choisit q_1 t.q. $\tau_1 \leq J(\nu)^{\downarrow}$ ce qui est possible puisque $\nu \in \{\tau_{min}(C) \leq J^{\downarrow}\}$. Ensuite :*
   - *Si $\tau_1 \geq {}^{\uparrow}J(\nu)$, alors comme précédemment on peut considérer $r = 0$ et prouver le résultat.*
   - *Sinon $\tau_1 < {}^{\uparrow}J(\nu)$.*

- *Si* $\mathsf{firable}(C) = \emptyset$*, alors la seule exécution maximale partant de* q_1 *est* $\rho = q_1$*. Pour* $r = {}^{\uparrow}J(\nu) - \tau_1 > 0$*,* $\rho^*(r) \models \psi$ *et* $\forall r' < r$*,* $\rho^*(r') \models \varphi$ *car nous restons dans la même classe* C*, et* $r \in J(\nu) - \tau_1$*, donc* $\nu \in F^{\phi}(C)$*.*
- *Sinon,* $\exists t \in \mathsf{firable}(C)$ *t.q.* $C' = (M', D') = \mathsf{succ}(C, t)$*, et d'après la formule* $F^{\phi}_{EU}(C)$ $\nu \in \{\tau_{max}(C') \geq {}^{\uparrow}J\} \wedge D'_{|Par}$*.*
  Puisque $\nu \in D'_{|Par}$*, la classe* $[\![C']\!]_\nu$ *est accessible ; il existe donc :* $\rho \in \pi(q_1)$ *et* $q_2 \in [\![C']\!]_\nu$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$*. On choisit* q_2 *t.q.* $\tau_2 = \tau_{max}(C')(\nu)$ *; alors* $\tau_2 \geq {}^{\uparrow}J(\nu)$ *et* $d_1 = \tau_2 - \tau_1$*.*
  On considère maintenant $r = \min(d_1, J(\nu)^{\downarrow} - \tau_1)$*. Nous avons* $\rho^*(r) \models \psi$ *et* $\forall r' < r$*,* $\rho^*(r') \models \varphi$ *car nous restons dans la classe* C*. Et naturellement, par construction,* $r \in J(\nu) - \tau_1$*. Donc au final* $\nu \in F^{\phi}(C)$*.*

3. *Dans le dernier cas* ν *vérifie le troisième terme de la disjonction. Il existe alors* $t \in \mathsf{firable}(C)$ *t.q.* $\nu \in F^{\phi}_{EU}(\mathsf{succ}(C, t))$*. On suppose ici que le résultat calculé par l'algorithme pour le successeur est correct, i.e.* $\exists q_2 \in [\![C']\!]_\nu$*,* $\exists \rho' \in \pi(q_2)$*,* $\exists r_1 \in J(\nu) - \tau_2$*, t.q.* $\rho'^*(r_1) \models \psi$ *et* $\forall r'_1 < r_1$ $\rho'^*(r'_1) \models \varphi$*.*
   Dans ce cas il existe $q_1 \in [\![C]\!]_\nu$ *et* $\rho \in \pi(q_1)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \rho'$ *et* $d_1 = \tau_2 - \tau_1$*. Pour* $r = d_1 + r_1$ *nous avons alors* $\rho^*(r) \models \psi$ *et* $\forall r' < r$*,* $\rho^*(r') \models \varphi$*. De plus,* $r \leq \tau_2 - \tau_1 + J(\nu)^{\downarrow} - \tau_2 = J(\nu)^{\downarrow} - \tau_1$ *et* $r \geq \tau_2 - \tau_1 + {}^{\uparrow}J(\nu) - \tau_2 = {}^{\uparrow}J(\nu) - \tau_1$*, donc* $r \in J(\nu) - \tau_1$*. Donc au final* $\nu \in F^{\phi}(C)$*.*

Nous avons prouvé que lorsque ν *vérifie la formule calculée par l'algorithme, quelque soit le cas la valuation est correcte, ce qui prouve la correction du semi-algorithme EU.*

Preuve A.2 (Théorème A.1 (complétude)) *Nous considérons maintenant une solution du problème du model-checking paramétré :* $\nu \in F^{\phi}(C)$*. Nous montrons que cette solution vérifie aussi la formule calculée par l'algorithme, i.e.* $\nu \in F^{\phi}_{EU}(C)$*.*
Nous prouvons le résultat par récurrence sur le nombre i *de transitions qui doivent être tirées pour prouver la formule* ϕ*. L'hypothèse de récurrence est*
$\mathbf{H_i}$ *: Soit* $C = (M, D) \in \mathcal{G}_c(\mathcal{N})$ *et* $\nu \in D_{|Par}$*. Si* $\exists q_1 \in [\![C]\!]_\nu$*,* $q_1 \models [\![\phi[J - \tau_1]]\!]_\nu$*, i.e..* $\exists \rho \in \pi(q_1)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t_1)} q_2 \xrightarrow{(d_2,t_2)} \cdots$ *et* $\exists r \in J(\nu) - \tau_1$ *t.q.* $\rho^*(r) \models \psi$ *et* $\forall r' < r$*,* $M, \rho^*(r') \models \varphi$*, avec* $r = \sum_{j=0}^{i} d_j + \delta$*, et* $d_0 = 0$ *et* $\delta < d_{i+1}$*, alors* $\nu \in F^{\phi}_{EU}(C)$*.*

$\mathbf{H_0}$ *: nous prouvons le résultat pour* $i = 0$

- *Si* $M \models \psi$ *et* $M \not\models \varphi$*. Alors, nécessairement* $r = 0$*, et donc* $0 \in [{}^{\uparrow}J(\nu) - \tau_1, J(\nu)^{\downarrow} - \tau_1]$*. De plus,* $\tau_1 \geq \tau_{min}(C)(\nu)$*, ce qui prouve que* $\tau_{min}(C)(\nu) \leq J(\nu)^{\downarrow}$ *et donc que* $\nu \in \{\tau_{min}(C) \leq J^{\downarrow}\}$*. De manière similaire* $\tau_1 \leq \tau_{max}(C)(\nu)$*, et donc* $\nu \in \{\tau_{max}(C) \geq {}^{\uparrow}J\}$*. Nous avons alors prouvé que* ν *vérifie le premier membre de la disjonction dans* $F^{\phi}_{EU}(C)$*.*
- *Sinon* $M \models \psi$ *et* $M \models \varphi$*. Il existe alors* $\rho = q_1 \xrightarrow{d_1} q_1 + d_1 \rightarrow \ldots$*, et* $0 \leq r < d_1$ *t.q.* $\rho^*(r) \models \psi$*. Si* $\rho = q_1$ *est maximale, alors* $\mathsf{firable}(C) = \emptyset$ *et puisque* $\tau_{min}(C)(\nu) \leq \tau_1 \leq r + \tau_1 \leq J(\nu)^{\downarrow}$*, on prouve directement que* $\nu \in F^{\phi}_{EU}(C)$*.*
  Sinon, $\exists t$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \cdots$*, et donc* $t \in \mathsf{firable}(C)$*, et pour* $C' = \mathsf{succ}(C, t)$ *nous avons* $\nu \in D'_{|Par}$*. Dans ce cas* $\tau_{max}(C')(\nu) \geq \tau_2$ *et* $\tau_2 = \tau_1 + d_1$*. En conséquence* $\tau_{max}(C')(\nu) \geq \tau_1 + d_1 > \tau_1 + r$*. Finalement, puisque* $r \in J(\nu) - \tau_1$ *nous obtenons que* $\tau_{max}(C')(\nu) \geq J(\nu)^{\uparrow}$ *et en donc* $\nu \in F^{\phi}_{EU}(C)$*.*

Nous avons prouvé l'hypothèse de récurrence pour $i = 0$. Nous prouvons maintenant la récurrence.

$\mathbf{H_i} \Rightarrow \mathbf{H_{i+1}}$ *: on suppose H_{i+1}, i.e. $\exists q_1 \in [\![C]\!]_\nu$, $\exists \rho \in \pi(q_1)$, $\exists r \in J - \tau_1$ t.q. $\rho^*(r) \models \psi$ et $\forall r' < r,\ \rho^*(r') \models \varphi$, avec $r = \sum_{j=1}^{i+1} d_j + \delta$ (on élimine le cas $j = 0$ car $i + 1 \geq 1$ et $d_0 = 0$).*
Puisque $i + 1 > 0$, $\exists t \in \mathsf{firable}(C)$ t.q. $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$. Soit alors $C' = (M', D') = \mathsf{succ}(C, t)$. $\nu \in D'_{|Par}$ puisque la classe est accessible. Nous avons $q_2 \in [\![C']\!]_\nu$ et $\tau_2 = \tau_1 + d_1$.
Nous considérons la sous-séquence de ρ commençant en q_2. Soit $r_1 = r - d_1 = \sum_{j=2}^{i+1} d_j + \delta$. Puisque $r \in J(\nu) - \tau_1$, on déduit que $r_1 \in J(\nu) - \tau_2$. De plus, $\rho'^*(r_1) = \rho^*(r)$, et donc $\rho'^*(r_1) \models \psi$. De manière similaire $\forall r'_1 < r_1,\ \rho'^*(r'_1) \models \varphi$. Nous avons alors vérifié toutes les hypothèses de H_i, et nous pouvons donc déduire par récurrence que $\nu \in F^\phi_{EU}(C')$.
Nous prouvons donc que ν vérifie le dernier membre de la disjonction dans $F^\phi_{EU}(C)$. Nous avons donc prouvé par récurrence la complétude du semi-algorithme EU.

A.2 Semi-algorithme AU

Théorème A.2 *Soit $\mathcal{N}$ un PITPN et $\phi = \forall \varphi\ \mathcal{U}_J\ \psi$ une formule PTCTL telle que $J \in \mathcal{J}(Par)$. Étant donnée une classe d'états paramétrée étendue $C = (M, D)$ du graphe des classes d'états $\mathcal{G}_c(\mathcal{N})$, la correction et la complétude du semi-algorithme AU sont exprimées par :*

$$F^\phi(C) = F^\phi_{AU}(C)$$

Preuve A.3 (Théorème A.2 (correction)) *Nous considérons une valuation qui vérifie la formule logique calculée par l'algorithme : $\nu \in F^\phi_{AU}(C)$. Nous montrons que cette valuation vérifie bien le problème du model-checking paramétré, c'est-à-dire que : $\nu \in F^\phi(C) = \{\nu \in D_{|Par} \mid \forall q_1 \in [\![C]\!]_\nu,\ q_1 \models [\![\phi[J - \tau_1]]\!]_\nu\}$.*

De la formule $F^\phi_{AU}(C)$, nous déduisons directement que $\nu \in D_{|Par}$, ce qui implique que $[\![C]\!]_\nu \neq \emptyset$. On note aussi que $\nu \in \{\tau_{max}(C) \leq J^\downarrow\}$. Il existe ensuite 3 manières de vérifier la formule disjonctive :

1. *Si ν vérifie le premier terme de la disjonction (i.e. $M \models \psi\ \wedge \{\tau_{min}(C) \geq {}^\uparrow J\}$).*
   Alors, $\forall q_1 \in [\![C]\!]_\nu,\ \tau_1 \in J(\nu)$, parce que d'une part $\tau_1 \geq \tau_{min}(C)(\nu) \geq J(\nu)^\uparrow$, et d'autre part $\tau_1 \leq \tau_{max}(C)(\nu) \leq J(\nu)^\downarrow$. En conséquence $\forall \rho \in \pi(q_1)$, pour $r = 0$, $\rho^*(r) \models \psi$ et $r \in J(\nu) - \tau_1$. Et donc $\nu \in F^\phi(C)$.
2. *Si ν vérifie le second terme de la disjonction, $\forall q_1 \in [\![C]\!]_\nu$, comme précédemment $\tau_1 \leq J(\nu)^\downarrow$, et alors soit $\tau_1 \geq J(\nu)^\uparrow$, auquel cas la preuve précédente s'applique à nouveau, ou bien $J(\nu)^\uparrow - \tau_1 > 0$ et dans ce cas :*
   - *Si $\mathsf{firable}(C) = \emptyset$, alors la seule exécution maximale partant de q_1 est $\rho = q_1$. Pour $r = {}^\uparrow J(\nu) - \tau_1 > 0$, $\rho^*(r) \models \psi$ et $\forall r' < r,\ \rho^*(r') \models \varphi$ car nous restons dans la même classe C, et $r \in J(\nu) - \tau_1$, donc $\nu \in F^\phi(C)$.*
   - *Sinon, $\forall \rho \in \pi(q_1)$, $\exists t \in \mathsf{firable}(C)$ t.q. $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$ et $d_1 = \tau_2 - \tau_1$. Soit $C' = (M', D') = \mathsf{succ}(C, t)$, et $D'' = D' \wedge \{\theta_c > -{}^\uparrow J\}$ et $C'' = (M', D'')$. De la formule logique on déduit que $\nu \in (F^\phi_{AU}(C'') \vee \neg D''_{|Par})$ et $\nu \in D'_{|Par}$ puisque dans ce cas la classe $[\![C']\!]_\nu$ est accessible.*

- *Si* $\tau_2 \geq J(\nu)^{\uparrow}$, *alors pour* $r = \min(d_1, J(\nu)^{\downarrow} - \tau_1)$, $\rho^*(r) \models \psi$, $\forall r' < r, \rho^*(r') \models \phi$ *parce que nous restons dans la classe* C. *De plus* $r \in J(\nu) - \tau_1$ *parce que soit* $r = J(\nu)^{\downarrow} - \tau_1$, *ou bien* $r = d_1$ *et alors* $r \leq J(\nu)^{\downarrow} - \tau_1$ *et* $r = \tau_2 - \tau_1 \geq J(\nu)^{\uparrow} - \tau_1$.
- *Sinon, le dernier cas est lorsque* $\tau_2 < J(\nu)^{\uparrow}$. *Il existe alors* ν' *t.q.* $(\nu|\nu') \in (D' \wedge \{\tau_c < J^{\uparrow}\})$, *ce qui signifie que* $(\nu|\nu') \in D''$ *ou encore que* $\nu \in D''_{|Par}$. *Par conséquent, nous pouvons en déduire par la formule* $F^{\phi}_{AU}(C)$ *que* $\nu \in F^{\phi}_{AU}(C'')$. *En supposant le résultat correct, nous obtenons que* $\forall q'_2 \in [\![C'']\!]_\nu$, $\forall \rho' \in \pi(q'_2)$, $\exists r_1 \in J(\nu) - \tau'_2$ *t.q.* $\rho'^*(r_1) \models \psi$, $\forall r'_1 < r_1$ $\rho'^*(r'_1) \models \varphi$. *Cela est vrai en particulier pour* $q_2 \in [\![C'']\!]_\nu$. *Maintenant, pour* $r = d_1 + r_1$ *nous prouvons le résultat pour* ρ.

3. *Finalement, dans le dernier cas* ν *vérifie le troisième terme de la disjonction. Alors* $\forall q_1 \in [\![C]\!]_\nu$, $\forall \rho \in \pi(q_1)$, $\exists t \in \mathsf{firable}(C)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \dots$. *Cela signifie que la classe* $[\![C']\!]_\nu$ *est accessible, et donc que* $\nu \in D'_{|Par}$. *Alors, par déduction nous obtenons que* $\nu \in F^{\phi}_{AU}(C')$. *On suppose à nouveau le résultat correct, et l'on montre qu'il existe* r_1 *tel que pour* $r = d_1 + r_1$ *nous prouvons que* $\nu \in F^{\phi}(C)$.

Nous avons prouvé que dans tous les cas le semi-algorithme AU *est correct.*

Preuve A.4 (Théorème A.2 (complétude)) *Nous considérons maintenant une solution du problème du model-checking paramétré :* $\nu \in F^{\phi}(C)$. *Nous montrons que cette solution vérifie aussi la formule calculée par l'algorithme, i.e.* $\nu \in F^{\phi}_{AU}(C)$.
Nous prouvons le résultat par récurrence sur le nombre maximum n *de transitions qui sont tirées pour prouver la formule* ϕ. *L'hypothèse de récurrence est*
$\mathbf{H_n}$ *: Soit* $C = (M, D) \in \mathcal{G}_c(\mathcal{N})$ *et* $\nu \in D_{|Par}$. *Si* $\forall q_1 \in [\![C]\!]_\nu$, $q_1 \models [\![\phi[J - \tau_1]]\!]_\nu$, *i.e.* $\forall \rho \in \pi(q_1)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t_1)} q_2 \xrightarrow{(d_2,t_2)} \cdots$, $\exists r \in J(\nu) - \tau_1$ *t.q.* $\rho^*(r) \models \psi$ *et* $\forall r' < r$, $\rho^*(r') \models \varphi$, *avec* $r = \sum_{j=0}^{i} d_j + \delta$ *et* $d_0 = 0$ *et* $\delta < d_{i+1}$, *et le nombre de transitions tirées* i *est tel que* $i \leq n$, *alors* $\nu \in F^{\phi}_{AU}(C)$.

$\mathbf{H_0}$ *: nous prouvons l'hypothèse de récurrence au rang* $n = 0$. *Puisque nous avons* $\nu \in F^{\phi}(C)$ *et que aucune transitions n'est tirée cela implique que* $M \models \psi$. *Alors :*

- *Si* $M \not\models \varphi$, *nécessairement* $r = 0$, *et puisque* $r \in J(\nu) - \tau_1$ *cela implique que* $J(\nu)^{\uparrow} \leq \tau_1 \leq J(\nu)^{\downarrow}$. *Cela est vrai quelque soit* $q_1 \in [\![C]\!]_\nu$ *; en particulier pour* $q_1 \in [\![C]\!]_\nu$ *t.q.* $\tau_1 = \tau_{min}(C)(\nu)$. *Nous prouvons alors que* $\nu \in \{\tau_{min}(C) \geq J^{\uparrow}\}$. *De manière similaire, pour* $q_1 \in [\![C]\!]_\nu$ *t.q.* $\tau_1 = \tau_{max}(C)(\nu)$ *nous prouvons que* $\nu \in \{\tau_{max}(C) \leq J^{\downarrow}\}$. *En conséquence* ν *vérifie le premier terme de la disjonction dans* $F^{\phi}_{AU}(C)$.
- *Sinon* $M \models \varphi$.
  - *Si* $\mathsf{firable}(C) = \emptyset$, *alors comme précédemment pour* $q_1 \in [\![C]\!]_\nu$ *t.q.* $\tau_1 = \tau_{max}(C)(\nu)$, *nous avons* $0 \leq r \leq J(\nu)^{\downarrow} - \tau_1 \Rightarrow \tau_1 \leq J(\nu)^{\downarrow}$. *Nous prouvons donc que* $\nu \in \{\tau_{max}(C) \leq J(\nu)^{\downarrow}\}$. *C'est suffisant pour vérifier le second terme de la disjonction.*
  - *Sinon,* $\forall q_1 \in [\![C]\!]_\nu$, $\forall t \in \mathsf{firable}(C)$ *et* $\forall \rho \in \pi(q_1)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \dots$ *et* $d_1 = \tau_2 - \tau_1$, $\exists r < d_1$ *qui vérifie* ψ. *Soit* $C' = \mathsf{succ}(C, t)$ *et* $D'' = D' \wedge \{\tau_c < J^{\uparrow}\}$. $\forall q_2 \in [\![C']\!]_\nu$, *puisque* $J(\nu) - \tau_1 \leq r < \tau_2 - \tau_1$ *nous obtenons que* $J(\nu)^{\uparrow} < \tau_2$ *et par conséquent* $[\![D'']\!]_\nu = \emptyset$, *ce qui signifie que* $\nu \in \neg D''_{|Par}$. *Donc dans ce cas aussi* ν *vérifie le second terme de la disjonction dans* $F^{\phi}_{AU}(C)$.

L'hypothèse de récurrence est prouvée pour $n = 0$. *Nous prouvons maintenant la récurrence.*

$\mathbf{H_n \Rightarrow H_{n+1}}$ *: Nous supposons ici* H_{n+1} *i.e.* $\forall q_1 \in [\![C]\!]_\nu$, $\forall \rho \in \pi(q_1)$, $\exists r \in J(\nu) - \tau_1$ *t.q.* $\rho^*(r) \models \psi$ *et* $\forall r' < r,\ \rho^*(r') \models \varphi$, *avec* $r = \sum_{j=0}^{i} d_j + \delta$ *et* $i \leq n+1$.
Nécessairement, $0 \leq r \leq J(\nu)^{\downarrow} - \tau_1 \Rightarrow \tau_1 \leq J(\nu)^{\downarrow}$ *et donc pour* $\tau_1 = \tau_{max}(C)(\nu)$ *nous montrons que* $\nu \in \{\tau_{max}(C) \leq J^{\downarrow}\}$. *Alors :*

- *Si* $M \not\models \psi$ *alors* $i > 0$, *et donc* $\mathsf{firable}\,(C) \neq \emptyset$. $\forall t \in \mathsf{firable}\,(C)$ *; soit* $C' = \mathsf{succ}(C, t)$.
  - *Si* $\nu \notin D'_{|Par}$ *alors* $\nu \in \neg D'_{|Par}$.
  - *Sinon* $\forall q_2 \in [\![C']\!]_\nu$, $\forall \rho' \in \pi(q_2)$, $\exists \rho \in \pi(q_1)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$, *et à partir des hypothèses initiales* $\exists r = \sum_{j=1}^{i} d_j + \delta$ *tel que* $\rho^*(r)$ *prouve* ψ. *Maintenant, soit* $r_1 = r - d_1$, $r_1 = \sum_{j=2}^{i} d_j + \delta$. $r1 \geq 0$ *puisque* $r \geq d_1$. *De plus* $r_1 = r - d_1 \leq J(\nu)^{\downarrow} - \tau_1 - d_1 = J(\nu)^{\downarrow} - \tau_2$ *et* $r_1 \geq J(\nu)^{\uparrow} - \tau_1 - d_1 = J(\nu)^{\uparrow} - \tau_2$, *ce qui prouve que* $r_1 \in J(\nu) - \tau_2$. *Évidemment,* $\rho'^*(r_2) \models \psi$ *et* $\forall r'_1 < r_1,\ \rho'^*(r'_1) \models \varphi$. *Finalement, le nombre de transitions tirées est* $i - 1 \leq n$. *Nous avons vérifié toutes les hypothèses de* H_n *et nous pouvons donc appliquer la récurrence ; nous obtenons que* $\nu \in F^{\phi}_{AU}(C')$.

  Donc en résumé pour chaque transition tirable, ν *vérifie le dernier terme de la disjonction dans* $F^{\phi}_{AU}(C)$.
- *Sinon* $M \models \psi$. *Si* $\mathsf{firable}\,(C) = \emptyset$ *nous obtenons directement que* $\nu \in F^{\phi}_{AU}(C)$. *Sinon* $\forall t \in \mathsf{firable}\,(C)$, *soit* $C' = \mathsf{succ}(C, t)$, *et* $D'' = D' \wedge \{\theta_c > -J^{\uparrow}\}$ *et* $C'' = (M', D'')$.
  - *Si* $\nu \notin D''_{|Par}$ *alors* $\nu \in \neg D''_{|Par}$.
  - *Sinon* $\nu \in D''_{|Par}$ *et alors* $\forall q_2 \in [\![C'']\!]_\nu$, $\forall \rho' \in \pi(q_2)$, $\exists \rho \in \pi(q_1)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$, *et donc* $\exists r = \sum_{j=1}^{i} d_j + \delta$ *qui vérifie* ψ.
    Comme précédemment, soit $r_1 = r - d_1 = \sum_{j=2}^{i} d_j + \delta$. *Puisque* $r \in J(\nu) - \tau_1$, $r_1 \in J(\nu) - \tau_2$ *mais de plus, puisque* $q_2 \in [\![C'']\!]_\nu$ *cela signifie que* $J(\nu)^{\uparrow} - \tau_2 > 0$, *et donc que* $r_1 > 0$. *Les hypothèses de* H_n *peuvent être à nouveau vérifiées et nous prouvons donc par récurrence que* $\nu \in F^{\phi}_{AU}(C'')$.

  Dans ce dernier cas aussi, ν *vérifie le second terme de la formule disjonctive dans* $F^{\phi}_{AU}(C)$.

Finalement dans tous les cas $\nu \in F^{\phi}_{AU}(C)$ *ce qui prouve* $\mathbf{H_n \Rightarrow H_{n+1}}$, *et par là même la récurrence et la complétude du semi-algorithme AU.*

A.3 Semi-algorithme LT

Théorème A.3 *Soit* $\mathcal{N}$ *un PITPN et* $\phi = \varphi \rightsquigarrow_{J_r} \psi$ *une formule PTCTL telle que* $J_r \in \mathcal{J}(Par)$, *avec* $J_r = [0, b]$ *et* $b \in \mathbb{Q}^+ \cup Par$, *ou* $J_r = [0, \infty[$. *Étant donnée une classe d'états paramétrée étendue* $C = (M, D)$ *du graphe des classes d'états* $\mathcal{G}_c(\mathcal{N}_{LT})$, *la correction et la complétude du semi-algorithme LT sont exprimées par :*

$$F^{\phi}(C) = F^{\phi}_{LT}(C)$$

Preuve A.5 (Théorème A.3 (correction)) *Nous considérons une valuation qui vérifie la formule logique calculée par l'algorithme :* $\nu \in F^{\phi}_{AU}(C)$. *Nous montrons que cette valuation vérifie bien le problème du model-checking paramétré, c'est-à-dire que :* $\nu \in F^{\phi}(C)$.
$\forall q_1 \in [\![C]\!]_\nu,\ \forall \rho \in \pi(q_1)$ *:*

1. *Si* $M(P_{LT}) = 0$, *et si* $\mathsf{firable}\,(C) = \emptyset$ *nous prouvons directement le résultat puisque* $M(P_{LT})$ *est constamment nul. Sinon,* $\exists t \in \mathsf{firable}\,(C)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$.

Dans ce cas, $\forall 0 \leq r_2 \leq d_1$ *nous prouvons le résultat puisque* $M(P_{LT}) = 0$. *Pour les états suivant, nous avons* $\nu \in F^{\phi}_{LT}(C')$ *et de plus puisque* $M(P_{LT}) = 0$ *l'horloge a été réinitialisée dans la classe* C' *(et donc les deux définitions de* $F^{\phi}(C')$ *sont équivalentes). Nous pouvons déduire que* $\forall \rho' \in \pi(q_2),\ \forall r'_2 \geq 0,\ \rho'^*(r'_2) \models (M(P_{LT}) = 1) \Rightarrow \exists r'_3 \geq r'_2,\ \ t.q.\ r'_3 - r'_2 \leq J_r(\nu)^{\downarrow}$ *et* $\rho'^*(r'_3) \models \psi$. *Et donc* $\forall r_2 > d_1$ *nous pouvons prendre* $r'_2 = r_2 - d_1$ *et prouver le résultat puisqu'alors* $\rho^*(r_2) = \rho'^*(r'_2)$.

2. *Sinon* $M(P_{LT}) = 1$.
   - *Si* $M \models \psi \wedge\ \tau_{max}(C)(\nu) \leq J_r(\nu)^{\downarrow}$ *alors nous considérons* $r_1 = 0$ *et nous avons* $0 \leq J_r(\nu)^{\downarrow} - \tau_{max}(C)(\nu) \leq J_r(\nu)^{\downarrow} - \tau_1$ *et évidemment* $\rho^*(r_1) \models \psi$.
     Ensuite, si $\mathsf{firable}\,(C) = \emptyset$ *alors* $\rho = q_1$ *et* $\forall r_2$ *pour* $r_3 = r_2$ *on vérifie que* $0 \leq J_r(\nu)^{\downarrow}$ *et* $\rho^*(r_3) \models \psi$.
     Sinon $\nu \in F^{\phi}_{LT}(C')$ *et comme précédemment nous montrons que le résultat est également vrai* $\forall r_2 > d_1$ *(dans ce cas nous ne considérons que la seconde définition parce que* $M \models \psi \Rightarrow M'(P_{LT}) = 0$*).*
   - *Sinon,* $M \not\models \psi$ *et donc* $\mathsf{firable}\,(C) \neq \emptyset$ *:* $\exists t \in \mathsf{firable}\,(C)$ *t.q.* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$ *et* $d_1 = \tau_2 - \tau_1$. *Nous avons* $\nu \in F^{\phi}_{LT}(C')$ *et* $q_2 \in [\![C']\!]_{\nu}$, *et puisque* ψ *n'a pas été vérifié dans* C *nous avons* $M'(P_{LT}) = 1$.
     En supposant le résultat de $F^{\phi}_{LT}(C')$ *correct,* $\forall \rho' \in \pi(q_2)$ *:*
     - *Nous obtenons tout d'abord que* $\exists r'_1 \geq 0$ *t.q.* $r'_1 \leq J_r(\nu) - \tau_2$ *et* $\rho'^*(r'_1) \models \psi$. *Alors pour* $r_1 = r'_1 + d_1$ *nous avons* $r_1 \leq J_r(\nu) - \tau_2 + \tau_2 - \tau_1$ *et donc* $r_1 \leq J_r(\nu) - \tau_1$ *et* $\rho^*(r_1) = \rho'^*(r'_1)$. *Nous avons alors vérifié* ψ.
     - *Pour les autres états, i.e.* $\forall r_2 \geq r_1$, *nous déduisons que* $\forall r'_2 \geq r'_1,\ \rho'^*(r'_2) \models (M(P_{LT}) = 1) \Rightarrow \exists r'_3 \geq r'_2,\ t.q.\ r'_3 - r'_2 \leq J_r(\nu)^{\downarrow}$ *et* $\rho^*(r'_3) \models \psi$. *Nous considérons maintenant* $r'_2 = r_2 - d_1$. *Nous avons* $r'_2 \geq r'_1$ *et donc* $\exists r'_3$. *Nous considérons ensuite* $r_3 = r'_3 + d_1$ *et prouvons le résultat.*

Nous avons alors prouvé que dans tous les cas le semi-algorithme LT est correct.

Preuve A.6 (Théorème A.3 (complétude)) *Nous considérons maintenant une solution du problème du model-checking paramétré :* $\nu \in F^{\phi}(C)$. *Nous montrons que cette solution vérifie aussi la formule calculée par l'algorithme, i.e.* $\nu \in F^{\phi}_{LT}(C)$.
Considérons le deux cas suivants :

1. *Si* $M(P_{LT}) = 0$ *alors* $\nu \in F^{\phi}(C)$ *implique par définition que* $\forall q_1 \in [\![C]\!]_{\nu}$, $\forall \rho \in \pi(q_1)$, $\forall r_2 \geq 0$ *la formule est prouvée. Si* $\mathsf{firable}\,(C) = \emptyset$ *le résultat est directement prouvé. Sinon,* $\forall t \in \mathsf{firable}\,(C)$ *et* $C' = \mathsf{succ}(C,t)$. *Soit* $D'_{|Par} = \emptyset$ *; ou bien* $\forall q_2 \in [\![C']\!]_{\nu}$, *il existe* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$. *Maintenant* $\forall \rho' \in \pi(q_2)$ *et* $\forall r'_2 \geq 0$, *on obtient* $\rho'^*(r'_2) = \rho^*(r_2)$. *Nous avons donc vérifié la définition de* $F_{\phi}(C')$ *pour* $M(P_{LT}) = 0$, *et par conséquent on peut induire que* $\nu \in F^{\phi}_{LT}(C')$. *Ceci prouve que* $\nu \in F^{\phi}_{LT}(C)$.
2. *Sinon* $M(P_{LT}) = 1$. *Dans ce cas* $\exists r_1 \geq 0$ *tel que* $\rho^*(r_1) \models \psi$ *et* $r_1 \leq J_r(\nu)^{\downarrow} - \tau_1$. *Ceci est valable pour tout* q_1, *en particulier pour* q_1 *t.q.* $\tau_1 = \tau_{max}(C)(\nu)$, *ce qui prouve que* $\nu \in \{\tau_{max}(C) \leq J_r^{\downarrow}\}$.
   Alors, si $\mathsf{firable}\,(C) = \emptyset$ *nécessairement* $M \models \psi$ *(puisque dans ce cas* $\rho^*(r_1) \in C$*), ce qui prouve* $\nu \in F^{\phi}_{LT}(C)$.
   Sinon, $\forall t \in \mathsf{firable}\,(C)$, *soit* $C' = \mathsf{succ}(C,t)$. *Soit* $D'_{|Par} = \emptyset$ *; ou bien* $\forall q_2 \in [\![C']\!]_{\nu}$, *il existe* $\rho = q_1 \xrightarrow{(d_1,t)} q_2 \rightarrow \ldots$. *Alors* $\forall \rho' \in \pi(q_2)$ *:*

- *Si* $r_1 \leq d_1$*, alors* $M \models \psi$ *et comme pour le cas* $M(P_{LT}) = 0$ *nous montrons par induction que* $\nu \in F^{\phi}_{LT}(C')$*.*
- *Sinon* $r_1 > d_1$*. Donc* $\exists r'_1 = r_1 - d_1$ *t.q.* $r'_1 \leq J_r(\nu) - \tau_1 - (\tau_2 - \tau_1) \leq J_r(\nu) - \tau_2$ *et évidemment* $\rho'^*(r'_1) \models \psi$*. Ensuite, de par la définition de* $F^{\phi}(C)$*,* $\forall r'_2 \geq r'_1$ *nous pouvons choisir* $r_2 = r'_2 + d_1$ *t.q.* $\rho^*(r_2) = \rho'^*(r'_2)$*. Encore une fois ceci prouve que* $\nu \in F^{\phi}(C')$ *et par induction que* $\nu \in F^{\phi}_{LT}(C')$*.*

Nous avons donc montré que $\nu \in F^{\phi}_{LT}(C)$*.*

Dans les deux cas la complétude du semi-algorithme LT est prouvée.

Résumé : Les travaux présentés portent sur l'étude de méthodes de vérification paramétrée des systèmes temps réel. La motivation pour ces recherches est de proposer des méthodes formelles à appliquer sur des systèmes dont les spécifications ne sont pas encore complètes. Des paramètres sont donc introduits dans les modèles utilisés afin de donner des degrés de liberté à la modélisation. Le but est alors de guider la conception du système en déterminant des valeurs satisfaisantes pour les paramètres.

Nous nous sommes focalisés sur les paramètres temporels qui sont en général parmi les plus complexes à définir. Nous avons ainsi défini le modèle des réseaux de Petri à chronomètres paramétrés.

Dans une première approche, nous étendons les méthodes d'analyse classiquement utilisées dans les réseaux de Petri temporels. L'espace d'états du modèle paramétré est ainsi représenté par le graphe des classes d'états paramétrées. Cela nous permet de proposer des semi-algorithmes de model-checking paramétré avec lesquels nous vérifions des formules de logique TCTL paramétrées.

Dans une seconde approche, nous étudions les méthodes qui préservent le parallélisme des réseaux de Petri. L'intérêt est de limiter l'explosion combinatoire qui apparaît lors de l'analyse de systèmes distribués, en particulier avec des modèles paramétrés. Nous proposons pour cela une méthode de dépliage temporel paramétré. Ce dépliage est a priori infini, mais nous proposons de l'utiliser pour résoudre un problème de supervision. La construction du dépliage est alors guidée par des observations finies, et nous extrayons les explications de ces observations, ainsi que les contraintes sur les paramètres qu'elles induisent.

Mots-clés : méthodes formelles, paramètres, réseaux de Petri temporels, model-checking, dépliages.

Abstract : The research works presented concern the study of parameterized verification methods for real time systems. The aim is to propose formal methods that can be applied on systems whose specifications are not yet completely defined. To that end, parameters are introduced into the formal models in order to add some degrees of freedom. The goal is then to guide the conception of the system by computing the admissible values of the parameters.

We have focused on the temporal parameters that are in general among the most difficult to determine. It led us to define the model of parametric time Petri nets with stopwatches.

In a first approach, we extend the classical analysis methods used for time Petri nets. First, we define the parametric state-class graph to represent the state-space of a parametric model. Then, it allows us to develop semi-algorithms for parametric model-checking, with which we can verify parametric TCTL formulae.

The second approach we have followed is the study of methods that preserve the natural concurrency of Petri nets. The goal is to limit the combinatory explosion that appears when checking distributed systems, especially when parametric models are considered. To that end, we propose to compute the parametric time unfolding of the parametric net. This unfolding is in general infinite, but we propose as an application to solve a supervision problem. In this context, the construction of the unfolding is guided by finite observations, and we can extract the explanations of these observations, as well as the constraints on the parameters that they induce.

Keywords : formal methods, parameters, time Petri nets, model-checking, unfolding.