HABILITATION À DIRIGER LES RECHERCHES

Olivier H. Roux

IRCCyN / Université de Nantes

Vérification des réseaux de Petri temporels et à chronomètres

École Doctorale : STIM

Spécialité : Automatique et Informatique Appliquée

Directeur de recherche : Jean-Pierre Elloy

Soutenue publiquement le **14 décembre 2005** à l'IRCCyN

Jury

Président	Robert Valette	Directeur de Recherche CNRS au LAAS
Rapporteurs	Hassane Alla	Professeur à l'Université Joseph Fourier de Grenoble
	Claude Jard	Professeur à L'École Normale Supérieure de Cachan (Antenne de Bretagne)
	Guy Juanole	Professeur à l'Université Paul Sabatier de Toulouse
Examinateurs	Jean-Pierre Elloy	Professeur à l'École Centrale de Nantes
	Yvon Trinquet	Professeur à l'Université de Nantes

Table des matières

Avant propos

Les travaux de recherche présentés dans ce mémoire concernent l'analyse des systèmes temps réel en comparant et en utilisant la complémentarité des modèles *réseaux de Petri temporels* et *automates temporisés* ainsi que de leurs extensions basées sur les chronomètres.

Ce mémoire est composé de deux parties : la première partie présente une synthèse de mes activités et une présentation générale de mes travaux de recherche. La deuxième partie présente un exposé scientifique détaillé d'une sélection de mes travaux les plus récents.

Première partie

Synthèse des activités

Chapitre 1

Curriculum Vitæ

1.1 Un bref Curriculum Vitæ

Coordonnées professionnelles actuelles

Recherche

IRCCyN - UMR CNRS 6597
1 rue de la noë
BP 92101
44321 Nantes cedex 03

téléphone : 02 40 37 69 76
fax : 02 40 37 69 30
olivier-h.roux@irccyn.ec-nantes.fr

Enseignement

IUT de Nantes
3, rue du Maréchal Joffre
B.P. 34103
44041 Nantes Cedex 1

téléphone : 02 28 09 21 59

olivier.roux@univ-nantes.fr

Situation professionnelle depuis ma thèse

1994-1995	**ATER à l'Ecole Centrale de Nantes**. Département automatique.
1995-1998	**Maître de Conférences à l'IUT de Kourou (Guyane Française)**. Département Génie Electrique et Informatique industrielle (GEII).
1998-2005	**Maître de Conférences à l'IUT de Nantes**. Département Génie Electrique et Informatique industrielle (GEII).
2004-2005	**Congé pour recherche**. Un semestre à l'IRCCyN et un semestre à l'école polytechnique de Montréal.

Expérience pédagogique

1991-1995 **à l'École Centrale Nantes**. Enseignement de l'*automatique* et de l'*électronique* en 2^{eme} année et *systèmes temps réel* en 3^{eme} année.

1995-1998 **à l'IUT de Kourou**. Enseignement de l'*électronique analogique, microélectronique* et *automatismes industriels*.

1998-2005 **à l'IUT de Nantes**. Enseignement de l'*électronique, traitement du signal* et *automatismes industriels*. Responsable pédagogique de l'enseignement de l'électronique en deuxième année. En congé pour recherche en 2004/2005.

2002 **En DEA**. Cours de DEA (20h) sur *les systèmes temps réel et la vérification* à l'université d'Abomey-Calavi au Bénin.

2005 **En MASTER recherche**. Cours de 24h sur les *réseaux de Petri* du MASTER Recherche *Automatique et Systèmes de production* de Nantes.

Activités de recherche

1991-1994 **Doctorat de l'Ecole Centrale Nantes**, spécialité « Automatique et Informatique Appliquée. » Préparé au Laboratoire d'automatique de Nantes (LAN) au sein de l'Equipe *Temps Réel* et soutenu de 17 juin 1994.

1994-1995 **Membre de l'équipe *Temps Réel* du Laboratoire d'Automatique de Nantes (LAN).**
- Participation aux travaux du groupe de normalisation ISO/TC 184/SC 2/WG 4 : « Programming methods and languages for manipuling industrial robots ».
- Participation au travail du GDR CNRS Protocoles-Réseaux-Systèmes (P.R.S.) thème Temps Réel.

1995-1998 **Membre du Centre de Recherche Scientifique et Technique de Guyane (CRSTG)**. Participation à la création du CRSTG et au montage du dossier de demande de statut de *jeune équipe* obtenu en 1996.

1999-2005 **Membre de l'équipe *Systèmes Temps Réel* de l'IRCCyN.**
- Participation à l'ACI CORTOS - ACI débutée en septembre 2003
- Participation au projet européen EAST-EEA (2001-2004).
- Participation au GDR ARP thème STS jusqu'en 2001 puis STRQDS depuis 2002.
- de février à juillet 2005 : Chercheur invité à l'école Polytechnique de Montréal.

Responsabilités collectives

1997-1998 **Chef du département GEII de l'IUT de Kourou**. J'assurais également la direction des études ainsi que la responsabilité des stages industriels.

1996-1998 **Responsabilités au sein de l'université Antilles-Guyane :**
- Membre du conseil d'administration de l'Université Antilles-Guyane.
- Membre du conseil d'administration de l'IUT de Kourou.
- Membre du conseil de gestion du CRIUG (Centre de ressources informatiques universitaire de Guyane).
- Membre de la CS (commune) 60, 61 et 63 eme sections de l'université Antilles-Guyane.
- Rédaction de la proposition et mise en place de la nouvelle option *Automatismes et Systèmes* du département GEII de l'IUT de Kourou.

1998-2005 **Responsabilités à l'université de Nantes et à l'IRCCyN**.
- Membre suppléant de la CS 61 de l'université de Nantes (2001-2004).
- Membre élu du conseil de l'IRCCyN (2000-2004).
- Responsable du laboratoire d'électronique de l'IUT de Nantes (Organisation, gestion, achat du matériel).

1.2 Encadrements et rayonnement

En plus des permanents de l'IRCCyN avec lesquels je travaille régulièrement et des DEA que j'ai encadrés, plusieurs personnes ont collaboré aux travaux décrits dans ce mémoire et ont co-signé les articles référencés. Il s'agit de doctorants que j'encadre (ou ai encadrés) et de collaborations nationales et internationales.

1.2.1 Encadrements

Encadrements de thèses de doctorat :

- Didier Lime (thèse commencée en octobre 2001 et soutenue le 1^{er} décembre 2004) sur la *Vérification d'applications temps réel à l'aide de réseaux de Petri temporels étendus*
  Encadrement : Yvon Trinquet (5 %), Olivier H. Roux (95%).
  Financement : projet européen EAST.
  Jury : B. Berthomieu, G. Juanole, S. Haddad, O. H. Roux, J. Sifakis, Y. Trinquet
  Publications associées : une revue internationale (DEDS [LR05a]), une revue nationale (TSI [LR05b]) et 4 conférences internationales (dont RTSS [LR04] et ICATPN [RL04]).
  Devenir du doctorant : post doc à Aalborg au Danemark (dans l'équipe de Kim Larsen) en 2004/2005. Maître de conférence à l'École Centrale de Nantes depuis septembre 2005.

– Guillaume Gardey (thèse commencée en octobre 2002. Soutenance prévue en décembre 2005) sur la *vérification et le contrôle des réseaux de Petri temporels*
Encadrement : Olivier F. Roux (50 %), Olivier H. Roux (50%).
Financement : bourse « Ministère ».
Cadre : ACI CORTOS. Publications associées : une revue internationale (TPLP [GRR06]) et 3 conférences internationales [GRR03, GLMR05, GMR05]
Devenir du doctorant : ATER à Bordeaux (ENSEIRB / LABRI) depuis septembre 2005.

– Morgan Magnin (thèse commencée en octobre 2004) sur *l'étude des réseaux de Petri à chronomètres en considérant une sémantique à temps discret.*
Encadrement : Pierre Molinaro (50 %), Olivier H. Roux (50%).
Financement : bourse BDI CNRS.
Publications associées : 2 conférences internationales [GLMR05, MLR05].

Encadrements de DEA ou de MASTER recherche :

– Olivier Sinquin, *Étude des extensions temporelles des réseaux de Petri*, 1999/2000.

– Didier Lime, *Vérification de propriétés temporelles sur les réseaux de Petri T-temporels*, 2000/2001.

– Guillaume Gardey, *Graphe des régions d'un réseau de Petri temporel*, 2001/2002.

– Cédric Motsch, *Réseaux de Petri temporisés à priorité : définition et calcul de l'espace d'état*, 2002/2003. En co-encadrement avec Pierre Molinaro.

– Morgan Magnin, *Vérification de réseaux de Petri temporels étendus à l'aide de polyèdres*, 2003/2004. En coencadrement avec Didier Lime.

– Etienne Borde, *D'un ADL temps réel vers les réseaux de Petri temporels*, 2005/2006. En partenariat avec Dassault Aviation.

– Charlotte Seidner, *Etude d'une traduction préservant la bisimulation d'un modèle défini selon l'approche CORE vers les réseaux de Petri temporels*, 2005/2006. En coencadrement avec Morgan Magnin. En partenariat avec l'entreprise *Sodius*.

1.2.2 Collaborations scientifiques personnelles.

Collaborations nationales :

– avec Bernard Berthomieu et François Vernadat du LAAS sur l'étude de problèmes de décidabilité sur les réseaux de Petri temporels étendus avec des chronomètres (« stopwatches »). Notre collaboration a conduit aux publications [BLRV04, BLRV05].

– avec Béatrice Berard et Serge Haddad du LAMSADE sur la comparaison de l'expressivité des automates temporisés et des réseaux de Petri temporels. Notre collaboration a conduit

aux publications [BCH+05a, BCH+05b, BCH+05c].

Collaborations internationales :

– avec Hanifa Boucheneb et John Mullins (Laboratoire de Conception et de Réalisation des Applications Complexes (LCRAC) de l'école Polytechnique de Montréal) sur la vérification de propriétés TCTL sur les réseaux de Petri temporels et sur la non-interférence dans le cadre des modèles temporisés. Hanifa Boucheneb est venue une semaine à l'IRCCyN en mai 2004. J'ai effectué un séjour de 6 mois à Montréal (en tant qu'invité) de fevrier à juillet 2005 et John Mullins doit normalement passer un mois à l'IRCCyN pendant le premier semestre 2006. Notre collaboration a conduit à la publication [GMR05] et deux autres publications sont en cours d'écriture.

– avec Enrico Vicario (Université de Florence) sur les relations entre les scheduling-TPN et les preemptive-TPN. Nous avons cherché l'année dernière à démontrer l'équivalence de ces deux modèles. Cette collaboration est très informelle et s'effectue par des échanges de courriers électroniques. Enrico Vicario avait proposé d'inviter Didier Lime en post doc pendant un an à Florence (offre déclinée par Didier qui a préféré effectuer son post doc dans l'équipe de Kim Larsen au Danemark). Je souhaite reprendre cette collaboration cette année et espère pouvoir envoyer Morgan Magnin faire un séjour de quelques semaines dans l'équipe d'Enrico Vicario pendant le premier semestre 2006.

1.2.3 Relecture d'articles.

J'ai effectué des évaluations d'articles (« review ») pour plusieurs conférences (parmi lesquelles ICATPN, TACAS, QEST, FORMATS, CSP et MSR) et pour les revues suivantes :

– IEEE Transactions on Software Engineering,
– International Journal of Computers and Applications,
– Journal of Systems and Software,
– International Journal of Production Research,
– European Journal of Control,
– European Journal of Automation,
– Technique et Science Informatiques.

A titre indicatif, pour l'année 2005, j'ai effectué deux évaluations d'articles pour *IEEE Transactions on Software Engineering*, une pour *Journal of Systems and Software* et une pour *International Journal of Computers and Applications*.

1.3 Soutiens financiers et valorisation - Projets et contrats

Les déplacements liés à ces collaborations, à mes travaux ainsi que certains salaires des doctorants ont été possibles grâce à trois projets (un projet français CORTOS, un projet européen EAST et un projet Canadien) auxquels s'ajoute un contrat avec Dassault Aviation :

1) Contrat *Dassault Aviation*. Dassault Aviation a pris contact avec moi en septembre 2004 pour une évaluation de l'outil Roméo. A l'issue de cette évaluation, nous avons signé avec Dassault Aviation un contrat de 12000 euros (pour l'année 2005) dont le but est, à partir d'une formalisation dans un langage de description d'architecture (AADL) de la gestion d'alimentation d'un système avionique, de faire une traduction de cette formalisation en réseaux de Petri temporels afin d'en extraire un premier jeu de règles de traduction. L'objectif est ainsi d'étudier la faisabilité d'une traduction automatique vers l'outil Roméo puis à terme de prouver formellement cette traduction et de l'implémenter.

2) L'ACI CORTOS : Control and Observation of Real-Time Open Systems. CORTOS est une ACI « Sécurité Informatique » et a débuté en septembre 2003. Elle implique quatre chercheurs de l'IRCCyN (dont moi-même) ainsi que des chercheurs du LSV (Cachan) et de VERIMAG (Grenoble). Son but est d'intégrer des aspects temps réel dans l'observation des systèmes ayant certains comportements masqués, de prendre en compte un temps non discret, et d'exprimer la correction du contrôle. La responsable du projet est Patricia Bouyer (LSV) et les deux responsables au sein de l'IRCCyN et de VERIMAG sont respectivement Franck Cassez et Stavros Tripakis. J'ai été sollicité pour ce projet en raison de la connotation à la fois *informatique* et *automatique* de mes activités sur le model-checking ; le problème du contrôle étant, dans ce cadre, à la frontière de ces deux thématiques. Je participe principalement à l'aspect contrôle basé sur les réseaux de Petri temporels. Une partie importante de la thèse de Guillaume Gardey s'effectue dans le cadre de cette ACI.

3) Le projet européen ITEA EAST-EEA (2001-2004) qui implique les constructeurs automobiles (français et allemands, principalement), les équipementiers et des universitaires, et traite de l'architecture électronique embarquée. Son objectif consiste à fournir une architecture électronique embarquée en proposant la définition d'une architecture ouverte qui réalise l'interopérabilité entre les composants matériels et les applications logicielles, le tout dans un environnement distribué. Ce projet au niveau de l'IRCCyN a été conduit du point de vue scientifique et financier par Yvon Trinquet. J'ai participé aux travaux du sous-groupe WP3 : tâche WT3.4 sur la vérification et la validation. Le travail de cette tâche (WT3.4) a principalement consisté à définir un langage commun entre industriels et chercheurs académiques sous la forme d'un glossaire précis et détaillé et à identifier les techniques et outils de vérification adaptés et applicables au contexte dicté par l'industrie automobile concernant les trois points suivants : i) *Model-checking / Analyse statique*, ii) *Simulation / Analyse dynamique* et iii) *Validation temps réel / Analyse des temps d'exécution*.

4) Enfin mon séjour à l'Ecole Polytechnique de Montréal de février à juillet 2005 a été pris en charge à 75% (sur 10000$) par un projet canadien (NSERC grant / Canadian Government) sur lequel travaille le laboratoire qui m'a invité à Montréal : le Laboratoire de Conception et de Réalisation des Applications Complexes.

1.4 Développement logiciel.

J'ai développé un outil logiciel appelé Roméo :

Description de l'outil Roméo .

Le logiciel Roméo est un atelier pour la saisie, la simulation, le calcul de l'espace d'états, la vérification et le contrôle des réseaux de Petri temporels et leur extension à chronomètres dédiée à la modélisation d'ordonnancements préemptifs. Ce logiciel est composé :

- une interface graphique écrite en tcl/tk,
- une librairie de simulation,
- des modules de calcul (écrits en C++).

Le logiciel fonctionne sus plates-formes *Windows*, *Linux* et *MacOS X*. La licence est une licence CeCILL et le logiciel est téléchargeable à l'adresse : http ://romeo.rts-software.org/

Animation de « l'atelier Roméo » J'anime les travaux autour de l'atelier logiciel Roméo. J'ai réalisé l le premier prototype de Roméo en 2000 et aujourd'hui, quatre personnes contribuent à son développement et à son rayonnement [GLMR05]. En effet, la plupart de nos travaux sur les réseaux de Petri ont conduit à des implémentations qui sont intégrées dans l'atelier Roméo. Ce logiciel a atteint aujourd'hui une certaine maturité ce qui a conduit à des relations académiques et industrielles.

- Roméo a récemment été intégré à l'outil PEP de l'université Ossietzky Oldenburg (Allemagne) qui permet aussi une exportation des fichiers PEP vers le format d'entrée de Roméo.
- Roméo a été évalué puis retenu (parmi 3 logiciels de model-checking) par Dassault Aviation pour la vérification de contraintes temporelles sur leurs systèmes avioniques (un contrat en 2005).
- Roméo est également utilisé par l'entreprise Sodius ce qui devrait conduire à une convention Cifre en 2006.

1.5 Séminaires

Invitation à des séminaires J'ai été sollicité pour effectuer des séminaires dans les groupes suivants :

- AS-CNRS sur les systèmes hybrides (AS 155 du RTP 24) ,
- AS-CNRS sur les *Systems On Chip* (SOC),
- Groupe de recherche sur le GRAFCET,
- ACI Chrono,
- GDR-CNRS STRQDS (Systèmes temps-réel et Qualité de Service).

Animation de séminaires J'organise les séminaires de l'équipe *Systèmes temps-réel* de l'IRCCyN depuis 2002 (La fréquence des séminaires est d'environ *un par mois* avec une pause à la période des soutenances de thèses).

J'invite ainsi les doctorants de notre équipes (et d'équipes connexes) à exposer leurs travaux et j'invite les permanents à nous présenter les thématiques nouvelles qu'ils peuvent aborder. De plus, j'invite (plus irrégulièrement) des chercheurs extérieurs à l'IRCCyN à participer à ce séminaire.

1.6 Publications

Revues internationales

[1] Franck Cassez and 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*, Elsevier, accepté en décembre 2005, à paraître.

[2] Didier Lime and Olivier (H.) Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Event Dynamic Systems - Theory and Applications (DEDS)*, accepté avec modifications mineures en 2004, accepté définitivement en 2005, à paraître.

[3] Guillaume Gardey, Olivier (H.) Roux, and Olivier (F.) Roux. State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems*, accepté en 2004, à paraître en 2006.

[4] Olivier (H.) Roux and Anne-Marie Déplanche. A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation (JESA)*, 36(7) :973–987, 2002.

[5] Olivier (H.) Roux and Pierre Molinaro. Deadlock detection and processing in Oreste. *European Journal of Automation (RAIRO - APII - JESA)*, 30(4) :519–542, 1996.

Revues nationales

[1] Didier Lime and Olivier (H.) Roux. Vérification formelle des systèmes temps réel avec ordonnancement préemptif. *Technique et Science Informatiques*, accepté en 2005, à paraître.

[2] Bernard Berthomieu, Didier Lime, Olivier H. Roux and François Vernadat. Problèmes d'accessibilité et Espaces d'états abstraits des réseaux de Petri Temporels à Chronomètres", *Journal européen des systèmes automatisés*, Numéro spécial en français sur la Modélisation des Systèmes Réactifs, 39(1-2-3) :223-238, 2005

[3] Olivier (H.) Roux and Pierre Molinaro. Detection en ligne des interblocages dans les systèmes répartis (on line distributed deadlock detection). *Technique et Science Informatiques*, 16(2) :243–263, 1997.

Conférences internationales.

Les conférences FSTTCS, FORMATS, CAV, RTSS, ICATPN et PNPM ont des taux d'acceptations inférieures à 50% (22% pour RTSS'04 et 28% pour ICATPN'04 par exemple).

Les conférences [1,3,4,5,9,12] ont été publiées dans les *Lecture Notes in Computer Science.*

[1] Beatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier (H.) Roux. When are timed automata weakly timed bisimilar to time Petri nets ? In *25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2005)*, Lecture Notes in Computer Science, Hyderabad, India, December 2005. Springer. à paraître.

[2] Guillaume Gardey, John Mullins, and Olivier (H.) Roux. Non-interference control synthesis for security timed automata. In *3rd International Workshop on Security Issues in Concurrency (SecCo'05)*, Electronic Notes in Theoretical Computer Science, San Francisco, USA, August 2005. Elsevier.

[3] Beatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier (H.) Roux. Comparison of the expressiveness of timed automata and time Petri nets. In *3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 05)*, Lecture Notes in Computer Science, Uppsala, Sweden, September 2005. Springer.

[4] Beatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier (H.) Roux. Comparison of different semantics for time Petri nets. In *Automated Technology for Verification and Analysis (ATVA'05)*, volume 3707 of *Lecture Notes in Computer Science*, Taiwan, October 2005. Springer.

[5] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier (H.) Roux. Roméo : A tool for analyzing time Petri nets. In *17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, Edinburgh, Scotland, July 2005. Springer.

[6] Morgan Magnin, Didier Lime, and 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)*, Electronic Notes in Theoretical Computer Science, Edinburgh, Scotland, UK, July 2005. Elsevier.

[7] Didier Lime and Olivier (H.) Roux. A translation based method for the timed analysis of scheduling extended time Petri nets. In *The 25th IEEE International Real-Time Systems Symposium, (RTSS'04)*, pages 187–196, Lisbon, Portugal, December 2004. IEEE Computer Society Press.

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

[9] Olivier (H.) Roux and Didier Lime. Time Petri nets with inhibitor hyperarcs. Formal semantics and state space computation. In *The 25th International Conference on Application and Theory of Petri Nets, (ICATPN'04)*, volume 3099 of *Lecture Notes in Computer Science*, pages 371–390, Bologna, Italy, June 2004. Springer.

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

[11] Didier Lime and Olivier (H.) Roux. Expressiveness and analysis of scheduling extended time Petri nets. In *5th IFAC International Conference on Fieldbus Systems and their Applications, (FET'03)*, pages 193–202, Aveiro, Portugal, July 2003. Elsevier Science.

[12] Guillaume Gardey, Olivier (H.) Roux, and Olivier (F.) Roux. A zone-based method for computing the state space of a time Petri net. In *In Formal Modeling and Analysis of Timed Systems, (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 246–259, Marseille, France, September 2003. Springer.

[13] Gilles Bernot, Franck Cassez, Jean-Paul Comet, Franck Delaplace, Céline Müller, Olivier Roux, and Olivier (H.) Roux. Semantics of Biological Regulatory Networks. In Vincent Danos and Cosimo Laneve, editors, *Workshop on Concurrent Models in Molecular Biology (BioConcur 2003)*, Electronic Notes in Theoretical Computer Science, Marseille (France), September 2003. Elsevier's ENTCS series.

[14] Pierre Molinaro, David Delfieu, and Olivier (H.) Roux. Improving the calculus of the marking graph of Petri net with bdd like structure. In *2002 IEEE international conference on systems, man and cybernetics (SMC 02)*, Tunisia, October 2002.

[15] Olivier (H.) Roux, David Delfieu, and Pierre Molinaro. Discrete time approach of time Petri net for real-time systems analysis. In *ETFA2001*, pages 197–204, Nice, France, October 2001. Volume 2, IEEE Catalog number : 01TH8597.

[16] David Delfieu, Pierre Molinaro, and Olivier (H.) Roux. Coupling binary decision diagrams with time Petri net. In *8th international conference on Real-Time and Embedded Systems*, pages 122–135, 2000.

[17] David Delfieu, Pierre Molinaro, and Olivier (H.) Roux. Analyzing temporal constraints with binary decision diagrams. In *25th IFAC Workshop on Real-Time Programming (WRTP'00)*, pages 131–136, Palma, Spain, 2000.

[18] Olivier (H.) Roux and Franck Roubaud. Shared Resources in Real-Time System. In *3th IEEE international conference on Industrial Applications (INDUSCON'98)*, São Paulo , Brésil, September 1998.

[19] Olivier (H.) Roux and Patrick Martineau . Deadlock Prevention in a Distributed Real-Time System. In *13th IFAC Workshop on Distributed Computer Control Systems (DCCS'95)*, Toulouse, September 1995.

[20] Olivier (H.) Roux and Pierre Molinaro. Specification and validation of the Oreste communication protocol (Spécification et validation du protocole de communication du langage Oreste). In *International Workshop On Principles Of Parallel Computing (OPOPAC'93)*, Lacanau, November 1993, pages 65-79, ISBN 2-86601-396-4.

[21] Olivier (H.) Roux and Pierre Molinaro. Oreste : a Reliable Reactive Real-Time Language. In *the 12th International Conference on Computer Safety, Reliability and Security (SAFECOMP'93)*, Poznan, Poland ; 27-29 October 1993, pages 302-313, Springer Verlag, ISBN 3-540-19838-5.

Conférences nationales.

[1] Bernard Berthomieu, Didier Lime, Olivier (H.) Roux, and Francois Vernadat. Problèmes d'accessibilité et espaces d'états abstraits des réseaux de Petri temporels à chronomètres. In *5ieme Colloque Francophone sur la Modélisation des Systèmes Réactifs, (MSR'05)*, Grenoble, France, October 2005.

[2] Franck Cassez and Olivier (H.) Roux. Traduction structurelle des réseaux de Petri temporels vers les automates temporisés. In *4ieme Colloque Francophone sur la Modélisation des Systèmes Réactifs, (MSR'03)*, Metz, France, October 2003.

[3] Olivier (H.) Roux and Anne-Marie Déplanche. Extension des réseaux de Petri t-temporels pour la modélisation de l'ordonnancement de tâches temps-réel. In *3ieme Colloque Francophone sur la Modélisation des Systèmes Réactifs, (MSR'01)*, pages 327–342, Toulouse, France, October 2001. Hermes Science.

[4] Olivier (H.) Roux and Pierre Molinaro. Mécanismes de communication et de synchronisation du langage Temps-Réel Oreste. In *JJCSIR*, Grenoble, April 1993 ; pages 5–10.

Chapitre 2

Présentation synthétique de mes travaux de recherche

2.1 Introduction

La classe des systèmes *temps réel* regroupe les programmes informatiques qui pilotent une application en réagissant à des stimuli reçus d'un environnement évolutif. Cela leur vaut également le nom de systèmes *réactifs*. Ces systèmes doivent réagir aux évolutions d'un procédé qu'ils contrôlent ou qu'ils suivent avec un temps de réponse suffisamment petit par rapport à sa dynamique interne. Ils sont soumis à des contraintes temporelles fortes. Il est donc important de s'assurer de leur correction non seulement fonctionnelle, mais aussi temporelle.

De plus, les applications temps réel étant en général critiques, il est important de détecter d'éventuelles erreurs le plus en amont possible dans la phase de conception afin de minimiser les coûts engendrés par leur correction. Plusieurs démarches de conception vont dans ce sens. Citons en deux : la première consiste, à partir d'un cahier des charges, à spécifier les propriétés de contrôle puis à modéliser le procédé à piloter ainsi qu'une solution du système de contrôle. La phase suivante est alors la validation de cette solution qui peut se faire par des techniques de tests, de démonstration automatique ou de vérification (...). Si le système n'est pas validé, il faut reprendre le problème en amont. Une deuxième démarche consiste, à partir d'un cahier des charges, à spécifier les propriétés de contrôle puis à modéliser le procédé à piloter et à synthétiser automatiquement une solution du système de contrôle garantissant la spécification. Si le contrôleur existe alors il faut en trouver une implémentation. Dans le cas contraire (ou en l'absence d'implémentation), il faut reprendre le problème en amont.

Mon activité de recherche s'inscrit dans ces deux démarches de conception par l'étude de méthodes formelles basées sur des modèles orientés états-transitions. Les modèles étudiés dans ce mémoire manipulent des actions considérées comme instantanées et identifiées comme pertinentes (qui représentent en général les événements associés aux commandes des actionneurs et aux mesures des capteurs du procédé) et des variables à valeurs réelles représentant l'écoulement de durées. Ils décrivent ainsi des sous classes des systèmes temps réel.

2.1.1 Les problèmes du *model-checking* et du *contrôle*.

Le cahier des charges d'une application informatique est en général formulé en langage naturel. Il est donc soumis à l'interprétation des rédacteurs de la spécification, ce qui est

source d'erreurs. Par ailleurs, pour des programmes d'une taille raisonnable, il est impossible, pour le concepteur, d'appréhender toutes les interactions entre les différents composants. Les méthodes formelles visent à offrir un cadre mathématique permettant de décrire de manière précise et stricte les systèmes et programmes que nous voulons construire. Dans ce cadre mathématique, le système est décrit par un système de transitions étiquetées (ou un modèle permettant d'abstraire un tel système comme les automates, les réseaux de Petri, les algèbres de processus...) et la spécification de la correction du système est décrite par des propriétés qui peuvent être exprimées sous la forme d'observateurs du modèle ou dans une logique particulière telle que les logiques temporelles linéaire (LTL) et arborescente (CTL) ou encore l'extension temporisée de cette dernière : TCTL (*Timed Computation Tree Logic*). Un exemple classique de propriété temps réel est la réponse bornée qui spécifie que lors de l'exécution du système, si une formule P_0 sur les états du système devient vraie alors une formule P_1 deviendra vraie avant n unités de temps.

Parmi les méthodes formelles, le model-checking est une procédure automatique qui permet de vérifier qu'un modèle d'un système satisfait une spécification décrite par une propriété. Cette procédure s'effectue par exploration des états du modèle grâce à des algorithmes tirant profit des modèles utilisés pour le système et pour la propriété.

La synthèse de contrôleur consiste, lorsque cela est possible, à construire un programme (le contrôleur) qui pilote les mécanismes d'interaction avec l'environnement (le procédé) de façon à garantir un fonctionnement sûr et correct. L'ensemble des actions du système est partitionné en deux sous ensembles : l'ensemble des actions contrôlables et l'ensemble des actions incontrôlables qui correspondent en général respectivement aux commandes des actionneurs et aux mesures des capteurs du procédé. Le contrôleur du procédé peut agir sur (et seulement sur) les actions contrôlables et le système résultant de l'adjonction du contrôleur est en boucle fermée (tandis que le procédé avant contrôle est dit en boucle ouverte).

Situons maintenant plus précisément le problème du contrôle par rapport à celui du model-checking. Dans le cas du model-checking, on considère un système déjà *fermé* S_{ferme} que l'on peut voir comme un procédé S auquel on a adjoint un contrôleur C. La propriété de correction attendue étant formalisée par une formule ϕ, le model-checking est un algorithme qui permet de répondre à la question « est-ce que S_{ferme} satisfait ϕ ? » c'est-à-dire « est-ce que S contrôlé par C satisfait ϕ ? ». Soit formellement :

$$\text{Est-ce que } S \parallel C \models \phi \quad ? \tag{2.1}$$

La problématique du contrôle est plus générale. Nous cherchons à *synthétiser* un *contrôleur* C pour S qui garantisse la propriété souhaitée ϕ sur le système fermé S_{ferme}. S est qualifié de *processus en boucle ouverte*. Le problème du *contrôle* consiste donc à décider si :

$$\text{Existe-t-il } C \text{ tel que } S \parallel C \models \phi \quad ? \tag{2.2}$$

Ce problème est généralement plus compliqué que celui du model-checking classique. Pour répondre au problème (2.2) il faut délimiter l'ensemble dans lequel on va chercher un contrôleur. Par exemple on peut chercher des contrôleurs qui sont à mémoire bornée (modélisables par un *automate fini*) ou non (*réseaux de Petri*...). De plus, le contrôleur construit doit être contraignant pour garantir les propriétés de sûreté, mais doit être suffisamment permissif pour ne pas réduire inutilement les comportements du système.

2.1.2 Les modèles temporisés.

Étant donnée une propriété que l'on veut vérifier (ou garantir par un contrôleur) sur un système (respectivement sur un procédé), un modèle peut être défini comme une abstraction du système telle que la valeur de vérité de la satisfaction de la propriété est la même pour le système et son modèle. Cela implique qu'un modèle n'est valable que pour un ensemble de propriétés et ne reflète pas complètement le comportement réel du système. Par ailleurs, il y a en général une opposition entre l'*expressivité* d'un modèle, c'est-à-dire sa capacité à représenter un nombre important de caractéristiques du système, et sa *simplicité* en termes de vérification ou de contrôle (décidabilité et complexité algorithmique).

D'un autre coté, choisir un modèle *suffisamment* expressif permet d'éviter le pessimisme d'un modèle surévaluant[1] le comportement réel du système. En effet, ce pessimisme peut d'une part, conduire à une propriété de sûreté décidée fausse sur le modèle alors qu'elle est vraie sur le système réel et d'autre part, conduire à un espace d'états infini alors que celui d'un modèle plus fin serait fini (ou aurait une abstraction finie). En particulier le modèle discret d'une application peut avoir un espace d'états non calculable alors que l'ajout des paramètres temporels de l'application, en restreignant les comportements, peut borner le nombre d'états discrets du modèle et permettre ainsi sa vérification.

De plus, dans certaines applications, le cahier des charges spécifie certaines propriétés dont la détermination nécessite la connaissance de durées séparant certaines transitions ou actions. Il est alors nécessaire de prendre en compte des caractéristiques temporelles autres que le temps logique (capturé par exemple par un modèle du type automate fini ou réseau de Petri) et de modéliser les dynamiques du procédé ou les réactivités du système de contrôle afin de vérifier des propriétés telles qu'un temps de réponse quantifié ou de faire apparaître des causes précises de dysfonctionnement.

Pour ces deux raisons (restreindre les comportements en supprimant ceux qui sont inexistants dans le système réel et vérifier des propriétés temporelles quantitatives) je m'intéresse à des modèles dit *temporisés* pour lesquels le temps est manipulé de manière explicite. Plus précisement, au sein des trois principales familles de modèles temporisés que sont les extensions au temps des algèbres de processus, des automates finis et des réseaux de Petri, je me suis tout d'abord porté vers les réseaux de Petri temporels en raison de leur puissance d'expression condensée du parallélisme et leur adéquation au problème de la modélisation des applications temps réel des systèmes embarqués. Ces travaux m'ont conduit à comparer les réseaux de Petri temporels aux automates temporisés afin d'en déduire des résultats de décidabilité et des méthodes de vérification et de contrôle.

Les automates temporisés. Les automates temporisés [AD94] (TA) étendent les automates finis classiques avec des horloges explicites. Dans ce modèle, à partir d'une localité, soit le temps s'écoule et les valeurs des horloges augmentent en conséquence, soit une transition discrète est franchie et l'action associée est effectuée avec éventuellement une remise à zéro de certaines horloges. Pour chaque transition est définie une *garde* qui contraint les horloges de l'automate. Cette garde doit être vérifiée pour que la transition puisse être franchie. Henzinger *et al.* étendent ce modèle à l'aide de contraintes sur les horloges, appelées *invariants* associés aux localités [HNSY94]. Le séjour dans une localité n'est possible que si l'invariant associé est vérifié ce qui permet de modéliser l'urgence.

[1] C'est à dire un modèle grossier ne permettant pas de distinguer des comportements fins dont certains sont en réalité impossibles.

Les réseaux de Petri et le temps. Les deux extensions temporelles principales des réseaux de Petri sont les réseaux de Petri *temporisés* [Ram74] et les réseaux de Petri *temporels* [Mer74]. Pour les premiers, le temps est représenté par des durées minimales (ou exactes dans le cas d'un fonctionnement « au plus tôt » du réseau) de franchissement des transitions. Pour les seconds, le temps prend la forme d'un intervalle contraignant les instants de tir des transitions. Par ailleurs le temps peut être associé aux transitions, aux places, aux jetons, aux arcs... Les classes de réseaux de Petri temporisés qui en résultent sont incluses dans les classes de réseaux de Petri temporels correspondantes [PT99]. Les principales classes de réseaux de Petri temporels sont les réseaux de Petri T-temporels [BD91], P-temporels [KDC96] et A-temporels [dFRA00] où un intervalle de temps est associé respectivement aux transitions, aux places et aux arcs. De plus, il existe deux sémantiques pour ces modèles : la sémantique faible pour laquelle les bornes temporelles supérieures peuvent être dépassées et la sémantique forte permettant la modélisation de l'urgence qui est une caractéristique essentielle pour les systèmes temps réel. Pour les réseaux de Petri A-temporels et P-temporels, la sémantique forte conduit à des jetons qui ne sont plus utilisables et qu'il faut faire *mourir* ce qui est parfois difficile à interpréter. Les réseaux de Petri T-temporels sont ainsi les plus utilisés pour la modélisation des systèmes temps réel et ce sont ceux qui nous étudions dans ce mémoire. Pour la suite, nous utilisons le terme *réseaux de Petri temporels* - que nous noterons TPN - pour désigner les réseaux de Petri T-temporels.

Enfin, les réseaux de Petri temporels (TPN) et les automates temporisés (TA) peuvent être étendus en mesurant l'écoulement du temps par des *chronomètres* à la place des *horloges* ce qui permet de modéliser des actions que l'on interrompt puis que l'on reprend plus tard. En effet, pour ces modèles, les dérivées par rapport au temps des variables peuvent prendre deux valeurs exprimant la progression (1) ou la suspension (0). Ces modèles entrent dans la classe des modèles hybrides. Ils permettent de modéliser les phases d'exécution et d'arrêt des activités d'un système ce qui permet de décrire finement des phases d'exécution et de suspension d'un programme, comme celles qui sont décidées par un ordonnanceur.

Temps dense ou temps discret ? Ces modèles formels temporisés permettent de modéliser l'évolution d'un système suite à des actions discrètes ou à l'écoulement du temps qui peut se faire de manière continue ou discrète. Le choix d'un temps dense ou d'un temps discret peut être motivé par les arguments (et contre arguments) suivants :

- les applications physiques (procédés) évoluent par rapport au temps physique qui est continu. La considération d'un temps discret conduit donc à une sous représentation des comportements réels du procédé. D'un autre coté, l'évolution du procédé n'est en général observée par le système de pilotage qu'à des instants particuliers (échantillonnage ou observations sporadiques) ;
- de son coté, le système de pilotage (de contrôle) est composé de tâches qui sont éxécutées sur un (ou plusieurs) processeur dont le temps *physique* est discret. L'utilisation d'un modèle en temps continu peut ainsi conduire à ajouter des comportements à ceux du système réel (l'envergure de cet ajout est fonction de la granularité temporelle du système de pilotage). Cependant, rien n'empêche avec un modèle en temps continu de discrétiser temporellement le comportement du système de pilotage.

Des arguments plus pragmatiques concernant l'efficacité ou l'existence d'algorithme de calcul de l'espace d'état peuvent aussi être considérés :

- le calcul de l'espace d'états des modèles temporisés en temps continu s'effectue par des

méthodes symboliques permettant le regroupement d'un ensemble (infini ou non) d'états en classes d'états évitant ainsi l'explosion que l'on obtient en considérant un temps discret et une énumération des états. Cependant, des structures de données récemment mises au point (basées sur des extensions des BDD) permettent d'appréhender les algorithmes de vérification de ces modèles à temps discret avec un soucis d'efficacité et ainsi de contenir ou compenser le problème de l'explosion du nombre d'état ;
– sur certains modèles en temps dense (à chronomètres par exemple), l'accessibilité n'est pas décidable ; il n'existe donc pas d'algorithme pour les vérifier ou calculer leur espace d'états mais seulement des semi-algorithmes. En revanche, l'utilisation d'un temps discret (avec des contraintes temporelles bornées) rend décidable le problème de l'accessibilité.

Mes travaux passés portent uniquement sur des modèles en temps dense cependant, concernant les modèles à chronomètres, je m'intéresse aussi depuis peu au temps discret pour des travaux qui ne sont pas décrits dans ce mémoire. Pour la suite de ce document, seules les sémantiques en temps dense sont considérées.

Je m'intéresse à deux classes de modèles : les modèles temporisés classiques et les modèles à chronomètres. Cette synthèse reprend cette distinction avec les sections 2.2 et 2.3.

Dans les deux cas, un premier paragraphe est consacré à des travaux théoriques liés à l'expressivité des modèles considérés et à la décidabilité de problèmes liés à la vérification et au contrôle de ces modèles. Ces travaux, menés en amont, permettent dans un premier temps d'identifier le modèle approprié à une application ou à un problème donné. Ensuite, ils permettent de déterminer si pour un problème particulier de model-checking ou de contrôle, on peut appliquer une méthode connue existante pour un autre modèle ou si l'on doit chercher une solution spécifique sous la forme d'un algorithme ou d'un semi-algorithme.

2.2 Travaux sur les modèles temporisés

2.2.1 Expressivité des TPN et des TA

Les deux principales classes d'expressivité pour les modèles temporisés sont l'expressivité en terme de bisimulation et celle en terme d'acceptation de langage temporisé (l'expressivité en terme de bisimulation implique l'expressivité en terme de langage). La recherche d'une classification précise de l'expressivité des modèles temporisés n'est pas qu'un simple jeu académique. Cela permet d'une part d'identifier le modèle approprié à un problème donné. D'autre part, cette recherche s'exprime souvent en terme de traduction d'un modèle vers un autre permettant de faire hériter au premier les résultats de décidabilité du deuxième en rapport avec la classe d'expressivité considérée. Par exemple si l'on prouve que la classe de modèle A est plus expressive que la classe de modèle B en terme de bisimulation par le biais d'une traduction de A vers B préservant la bisimulation et si un problème tel que la vérification de $TCTL$ est décidable sur la classe A alors, on obtient la décidabilité de ce même problème pour la classe B. De plus si la comparaison conduit à une égalité d'expressivité par une double traduction alors on peut aussi déduire des résultats de complexité algorithmique. De tels résultats nous indiquent ainsi l'existence et la complexité des l'algorithmes qu'il faut chercher.

Considérons les cas des réseaux de Petri temporels généraux (donc non bornés a priori) et des automates temporisés. Les réseaux de Petri temporels (TPN) permettent de modéliser di-

rectement des variables non bornées ce que ne peuvent pas faire les automates temporisés (TA) en absence de variable. D'un point de vue pratique, les TPN permettent une modélisation immédiate de problèmes concrets des applications temps réel : citons l'exemple du sémaphore dont on ne connait pas la borne (potentiellement inexistante), que l'on ne peut pas modéliser avec un automate temporisé alors qu'il se modélise trivialement par un réseau de Petri temporel à une place. D'un point de vue théorique, contrairement aux TA, les TPN ont le pouvoir d'expression d'une machine de Turing [JLL77] ce qui s'exprime en terme de décidabilité par le fait que l'accessibilité est décidable pour les TA ainsi que pour les TPN bornés et ne l'est pas pour les TPN généraux. La comparaison de ces deux modèles est restée sur ce constat pendant de nombreuses années sans établir si l'un des deux modèles est strictement plus expressif que l'autre (ou que l'une de ses sous-classes).

Une partie de mes travaux de ces dernières années a donc été consacrée à ce sujet.

Expressivité en terme de bisimulation.

Nous avons proposé avec Franck Cassez ([CR03, CR04]) une traduction structurelle des TPN vers les TA préservant la sémantique comportementale (bisimulation temporelle). Les classes des TPN considérées sont celles des TPN bornés ou non bornés avec ou sans contraintes strictes et ses différentes déclinaisons. La traduction, présentée dans le chapitre 5, consiste à traduire un TPN en un produit synchronisé d'automates temporisés avec variables[2]. Nous définissons pour cela un automate temporisé pour chaque transition du TPN. Cet automate temporisé possède une horloge représentant le temps de sensibilisation de la transition correspondante du TPN. Les états de l'automate donnent l'état de cette transition : sensibilisée, non-sensibilisée ou en train d'être tirée. Nous avons prouvé que notre traduction préserve le comportement du TPN initial dans le sens où la sémantique du TPN et sa traduction sont temporellement bisimilaires. Nous obtenons donc le résultat suivant :
Pour tout TPN borné, il existe un TA qui lui est temporellement bisimilaire.
Cela nous permet de déduire les résultats suivants :

- la classe des TPN bornés est d'expressivité inférieure ou égale à celle des TA en termes de langage et de bisimulation ;
- les TPN bornés héritent des résultats de décidabilité concernant le langage, établis sur les TA tels que : le langage vide est décidable sur les TPN bornés (notons que le test du langage vide permet de décider de l'accessibilité d'un état) ;
- les TPN bornés héritent des résultats de décidabilité liés à la bisimulation tels que : TCTL est décidable sur les TPN bornés.

De plus nous avons prouvé avec Béatrice Berard et Serge Haddad (du LAMSADE) dans [BCH$^+$05b] que la réciproque n'est pas vraie, c'est-à-dire :
Il existe un TA tel qu'il n'existe pas de TPN qui lui soit temporellement bisimilaire.

Ce résultat prouve que la classe des TA est strictement plus expressive en terme de bisimulation que la classe des TPN bornés. Nous avons alors identifié une caractérisation syntaxique de la sous-classe des TA (à contraintes larges uniquement et avec invariants ordonnés) équivalente aux TPN (à contraintes larges) [BCH$^+$05b] puis une caractérisation sémantique [BCH$^+$05c].

[2]Les variables utilisées dans les automates temporisés avec variables codent le marquage du réseau. Un automate temporisé avec variables bornés peut être encodé par un automate temporisé classique.

Expressivité en terme d'acceptation de langage temporisé.

Ce précédent résultat interroge sur l'expressivité en terme de langage temporisé qui est moins forte que celle en terme de bisimulation. Nous avons étudié cette question dans [BCH+05b] en proposant une traduction structurelle (syntaxique) des TA vers les TPN. Cette traduction consiste à traduire toutes les gardes du TA en un motif TPN tel qu'un marquage particulier de ce motif donne la valeur de vérité de la garde correspondante. Les différents motifs sont ensuite associés pour construire le TPN complet. Nous avons prouvé que le TA et sa traduction en TPN acceptaient le même langage temporisé. Le TPN obtenu est sauf (1-borné) ce qui nous permet de déduire les résultats suivants :

- les classes des TPN saufs et des TA sont d'expressivité égale en terme de langage temporisé ;
- les classes des TPN saufs et des TPN bornés sont d'expressivité égale en terme de langage temporisé ;
- les TPN bornés héritent des résultats d'indécidabilité concernant le langage, établis sur les TA tels que : le langage universel ainsi que l'inclusion de langage sont indécidables sur les TPN bornés.

Cette étude est détaillée dans le chapitre 5 de ce mémoire.

2.2.2 Vérification des TPN

La vérification d'une propriété particulière sur un modèle s'effectue en explorant l'espace d'états du modèle, totalement ou en partie. Pour les TPN et les TA, les horloges prennent leurs valeurs dans l'ensemble des réels (temps dense) ce qui implique que ces modèles ont en général un espace d'états infini. Il est donc nécessaire de recourir à une méthode *symbolique* afin de calculer une abstraction finie de l'espace d'états. L'approche *symbolique* consiste à partitionner l'espace d'états en un nombre fini de groupes d'états partageant une ou plusieurs propriétés « intéressantes », par exemple même marquage pour un réseau de Petri temporel ou accessibilité par la même séquence de transitions discrètes. Cela peut se traduire en : soit une relation d'équivalence $\mathcal{R}$ sur les états du système telle que deux états en relation par $\mathcal{R}$ partagent les propriétés P, calculer les classes d'équivalence définies par $\mathcal{R}$.

La méthode classique d'analyse des réseaux de Petri temporels est le *graphe des classes d'états* [BM83, BD91]. Dans cette méthode, l'espace d'états est partitionné selon la relation d'équivalence suivante : deux états sont en relation si et seulement si ils sont accessibles par une même séquence de transitions discrètes. En particulier, ils partagent le même marquage. Le graphe des classes d'états préserve les marquages et le langage discret du graphe des états accessibles. Il est donc adapté à la vérification de l'accessibilité de marquage et de propriétés LTL. Par contre, il ne préserve pas la structure de branchement du graphe des états ce qui rend impossible la vérification de propriétés CTL*. Pour remédier à ce problème, Yoneda et Ryuba proposent dans [YR98] un autre partitionnement de l'espace d'états en *classes atomiques*. Dans une classe atomique, tout état a un successeur dans chacune des classes atomiques obtenues par tir d'une transition. Les classes atomiques de Yoneda et Ryuba sont obtenues en découpant les classes d'états (définies de façon différente de [BD91]), par l'ajout de contraintes linéaires. Les auteurs prouvent que le graphe des classes atomiques permet la vérification de propriétés CTL*. Cependant cette technique n'est pas applicable aux réseaux pour lesquels les intervalles de temps ne sont pas bornés. Dans [BV03], Berthomieu et Vernadat proposent une construction alternative des classes d'états atomiques, qui fournit un graphe plus compact,

se calcule plus rapidement et est applicable aux réseaux dont les intervalles de temps ne sont pas forcément bornés.

Des outils existent (principalement Tina) pour calculer ces différents graphes des classes mais aucune méthode directe[3] n'a été proposée pour la vérification de propriétés temporelles quantitatives et aucun véritable outil de model-checking des TPN (c'est-à-dire permettant à partir du modèle d'un système S et d'une propriété φ, de décider $S \models \varphi$) n'était disponible avant notre outil Roméo.

En revanche, des méthodes et outils existent sur les automates temporisés pour la vérification de propriétés temporelles quantitatives. En particulier, l'outil Kronos permet la vérification de propriétés exprimées dans la logique temporelle temporisée TCTL et l'outil Uppaal implémente une méthode à la volée restreignant l'ensemble des propriétés TCTL considéré afin d'utiliser des algorithmes plus efficaces. La première méthode de calcul de l'espace d'états proposée sur les TA consiste à le partitionner en *régions* [AD94]. Les régions regroupent les états de l'automate temporisé selon une partition « géométrique » de l'espace des horloges. Le nombre de régions est fini mais très grand ce qui rend cette partition inutilisable pour la vérification mais la méthode est intéressante du point de vue théorique (pour prouver la décidabilité de problèmes tel que l'accessibilité d'états pour les automates temporisés). D'un point de vue pratique, la méthode la plus utilisée consiste à regrouper les régions en *zones* [LPY95] représentées par des *Difference Bound Matrices* (DBM). Les états à l'intérieur d'une zone sont tous les états accessibles entre le franchissement de deux transitions de l'automate. Cependant, des travaux récents ont montré les limitations des DBM. Patricia Bouyer [Bou02, Bou03] a prouvé que l'utilisation des DBM associée à un algorithme *en avant* conduit à une surapproximation de l'espace d'états : certains états sont alors dit atteignables alors qu'ils ne le sont pas.

Espace d'états - propriétés d'accessibilité.

Nous avons proposé dans [GRR03, GRR06] une approche efficace en-avant du calcul de l'espace d'états d'un TPN (dans le cadre du DEA et d'une partie de la thèse de Guillaume Gardey). Notre méthode est basée sur les zones et donc sur les DBM. Nous avons d'abord considéré la sous-classe des TPN pour laquelle les intervalles de tir des transitions sont bornés afin d'adapter l'algorithme utilisé pour les automates temporisés. Nous avons ensuite considéré la classe générale des TPN (autorisant l'infini en borne supérieure des intervalles de tir) pour laquelle un opérateur de surapproximation des zones peut être nécessaire pour assurer la terminaison des algorithmes. Nous avons prouvé qu'avec cette surapproximation notre algorithme termine et est exact vis-à-vis de l'accessibilité des marquages. L'implémentation de cette méthode dans notre outil Roméo s'est révélée très efficace ; elle ne permet toutefois la vérification de propriétés quantitatives que par l'intermédiaire d'observateurs qui traduisent la vérification de la propriété en un problème d'accessibilité d'un marquage particulier.

TCTL par TA intermédiaire.

La traduction des TPN vers les TA préservant la sémantique comportementale du TPN initial que nous avons décrite précédemment (dans la section 2.2.1 sur l'expressivité) dans un but théorique permet aussi d'un point de vue pratique la vérification de propriété TCTL. La démarche consiste à utiliser le TA obtenu par la traduction et de le vérifier avec les outils de

[3] C'est-à-dire sans utilisation d'observateur ou sans traduction vers les automates temporisés.

model-checking des TA tels qu'Uppaal ou Kronos mais cette méthode se révèle inefficace à cause du nombre d'horloge du TA obtenu (une par transition du réseau de Petri initial). Nous avons donc proposé dans [LR03b, LR05a] une autre traduction en TA, basée cette fois sur le calcul du graphe des classes d'états d'un réseau de Petri temporel borné. À partir de cette traduction, nous avons montré comment exprimer des propriétés $TCTL$ sur le réseau de Petri temporel et les vérifier efficacement, grâce à un algorithme de réduction du nombre d'horloges à la volée lors de la traduction, sur l'automate temporellement bisimilaire avec les outils Uppaal et Kronos. Comme nous le verrons plus loin cette méthode a été ensuite étendue pour les TPN à chronomètres.

TCTL à la volée.

Nous avons également étudié le model-checking de TCTL sur les réseaux de Petri temporels par des méthodes directes c'est-à-dire sans passer par un automate temporisé intermédiaire mais en s'inspirant en revanche des algorithmes et structures de données montrés comme étant les plus efficaces sur les TA. Les principaux résultats ont été obtenus lors de mon séjour à Montréal (de février à juillet 2005) pendant lequel nous avons travaillé sur sur la vérification des TPN avec Hanifa Boucheneb et Guillaume Gardey (qui est venu nous rejoindre six semaines suite à une invitation d'Hanifa Boucheneb).

Nous avons dans un premier temps proposé un TCTL pour les TPN : *TPN-TCTL*. D'un point de vue théorique, nous avons prouvé la décidabilité du model-checking de *TPN-TCTL* sur les TPN bornés par une preuve directe et avons montré que sa complexité est PSPACE. Nous nous sommes ensuite intéressés aux méthodes à la volée dont l'efficacité a été prouvée sur les automates temporisés avec l'outil Uppaal. Ces méthodes restreignent l'ensemble des propriétés TCTL dans le but d'utiliser des algorithmes plus efficaces mais les cas d'étude ont montré que la classe de propriétés vérifiables était suffisante pour les cas pratiques étudiés. Nous avons donc considéré un sous-ensemble de *TPN-TCTL* ($TPN\text{-}TCTL_{\mathcal{S}}$) et, à partir des algorithmes de [GRR03, GRR06], nous avons proposé un algorithme *en avant* de model-checking à la volée efficace utilisant une abstraction basée sur les zones. Afin d'accélérer la convergence, nous avons défini une approximation plus fine des zones conduisant à une abstraction plus compacte de l'espace d'états. Nous avons montré que notre abstraction est exacte vis-à-vis des propriétés de $TPN\text{-}TCTL_{\mathcal{S}}$. La méthode est implémentée et intégrée à notre outil Roméo [GLMR05]. Il est ainsi possible de vérifier des propriétés temps réel sur les TPN avec cet outil et de bénéficier de ses autres fonctionnalités : génération d'un contre-exemple lorsque la propriété est fausse, environnement de simulation ...

2.2.3 Contrôle des systèmes temporisés

Ce travail s'intègre dans la recherche menée au sein de l'ACI CORTOS. De plus une partie importante de la thèse de Guillaume Gardey (que j'encadre avec mon homonyme Olivier F. Roux) est sur ce sujet.

Contrôle des TPN.

Très peu de travaux existent sur la synthèse de contrôleur pour les réseaux de Petri temporels (en temps dense) et les rares travaux existants considèrent des TPN dont le réseau de Petri sous-jacent (non temporel) est sauf. Nous nous intéressons au problème du contrôle

dans les systèmes temporisés en considérant le modèle général des réseaux de Petri temporels (c'est-à-dire dont l'ensemble des marquages accessibles et les contraintes temporelles ne sont pas nécessairement bornés). L'ensemble des transitions est partitionné en un ensemble de transitions contrôlables et un ensemble de transitions incontrôlables. Nous définissons le contrôleur C comme une fonction sur l'ensemble des traces du réseau et nous considérons des propriétés exprimées sur les marquages du réseau. Si l'on appelle $Reach(\mathcal{N})$ l'ensemble des marquages accessibles d'un réseau $\mathcal{N}$, le problème à résoudre est : étant donné un réseau de Petri temporel $\mathcal{N}$ et un ensemble fini de marquage M, existe-t-il un contrôleur C tel que $Reach(\mathcal{N} \parallel C) \subseteq M$?

D'un point de vue pratique, un cas particulier très important de ce problème est : étant donné, un réseau de Petri temporel $\mathcal{N}$ et un entier naturel k, existe-t-il un contrôleur C tel que $\mathcal{N} \parallel C$ est k-borné ?

En effet, la bornitude d'un TPN étant indécidable, il est important de pouvoir prouver l'existence d'un contrôleur qui k-borne un réseau et de le synthétiser de manière à ce qu'il borne les éléments d'une application tels qu'un sémaphore, un tampon ou une pile.

Nous avons démontré que ces deux problèmes sont décidables et nous avons proposé une méthode pour répondre au problème de la synthèse du contrôleur C basée sur le calcul des prédécesseurs contrôlables tels que définis dans [MPS95, AMPS98]. Les théorèmes, les preuves et les algorithmes de synthèse de contrôleur sont détaillés dans [Gar05].

Contrôle de la non-interférence temporisé.

Mon activité sur le contrôle a conduit très récemment à des travaux en collaboration avec John Mullins de l'École Polytechnique de Montréal sur le contrôle de la non-intérference dans le contexte temporisé. Le problème est de contrôler les flux explicites ou implicites d'informations, par lesquels des observateurs pourraient obtenir des informations secrètes ou sensibles de manière malicieuse ou par inadvertance. Nous avons basé notre étude sur une modélisation par des automates temporisés dont l'ensemble des actions est partitionné en un ensemble d'actions de haut niveau (secrètes et privées) et un ensemble d'actions de bas niveau (publiques). Un système est non-interférent si un observateur de bas niveau ne peut pas déduire de son observation le comportement du haut niveau ce qui se traduit par : le comportement observé par un observateur de bas niveau (ne voyant pas les actions de haut niveau) est équivalent (par une relation d'équivalence qui peut être une égalité de langage, une simulation, une équivalence entre états, une cosimulation ou encore une bisimulation) au comportement que l'on obtient lorsque le haut niveau ne fait rien. Nous avons défini ces différentes classes de non-interférence temporisées et avons prouvé la décidabilité du problème de contrôle pour deux d'entre elles : l'une basée sur une équivalence d'états et l'autre sur une cosimulation en considérant que les actions contrôlables sont les actions de haut niveau.

Ces travaux émergents ont conduit à la publication [GMR05] et offrent de nombreuses perpectives sur lesquelles je compte travailler dans les mois à venir afin d'étudier plus profondément la notion de non-interférence dans un cadre temporisé et de proposer des algorithmes de vérification et de synthèse de contrôleur pour ce type de propriétés.

2.3 Travaux sur les modèles à chronomètres - vérification de systèmes temps réel avec ordonnancement préemptif

Notre motivation pour l'étude des modèles à chronomètres est principalement la vérification des applications temps réel avec ordonnancement préemptif. En effet, les modèles temporisés ne sont en général pas suffisants pour modéliser et vérifier ces applications. Dans les modèles temporisés classiques, le temps s'écoule de façon identique pour toutes les composantes du système, ce qui permet de modéliser des politiques d'ordonnancement non préemptives ou des tâches qui s'exécutent chacune sur un processeur différent. En revanche, ces modèles ne permettent pas de représenter les politiques d'ordonnancement préemptives où l'exécution d'une tâche est interrompue et reprise au même endroit un peu plus tard.

Mon activité sur ce sujet a débuté en 2001 lorsqu'avec Anne-Marie Deplanche, nous avons proposé une extension à chronomètres des TPN dédiée à la modélisation d'application temps réel avec ordonnancement préemptif. J'ai ensuite encadré la thèse de Didier Lime (soutenue en décembre 2004) sur l'étude de la vérification de ce modèle. Parallèlement à cela, nous avons collaboré avec Bernard Berthomieu et François Vernadat du LAAS pour établir des résultats (tels que l'indécidabilité de l'accessibilité) pour la classe des TPN à chronomètres. Aujourd'hui Pierre Molinaro et moi même, encadrons le travail de thèse de Morgan Magnin sur l'étude de méthodes de vérification efficaces sur ces modèles en considérant une sémantique à temps discret.

2.3.1 Modèles à chronomètres

Pour exprimer la suspension et la reprise d'actions, plusieurs modèles basés sur la notion de chronomètre (stopwatch) ont été proposés. Une extension des automates temporisés (TA), les automates à chronomètres (SWA), peut être définie comme une sous-classe des automates hybrides linéaires (LHA) pour laquelle les dérivées par rapport au temps des variables ne peuvent prendre que deux valeurs exprimant la progression (1) ou la suspension (0). Le problème de l'accessibilité est équivalent pour les SWA et les LHA [CL00]. Ce problème a été démontré indécidable pour LHA [ACH$^+$95a] et l'est donc pour SWA.

Plusieurs extensions des TPN ont été proposées pour répondre au problème de la modélisation de la suspension et de la reprise d'actions : les Scheduling-TPN [RD02] [LR03a], les Preemptive-TPN [BFSV04] et les TPN à hyperarcs inhibiteurs (IHTPN) [RL04]. Les deux premiers ajoutent des ressources et des priorités au modèle TPN, les IHTPN introduisent des arcs inhibiteurs qui contrôlent la progression des transitions.

Puisque tous étendent les TPN, le problème de l'accessibilité d'états est indécidable, mais en revanche on peut se poser la question de la décidabilité de l'accessibilité d'états lorsque le réseau est borné (qui est d'un intérêt pratique élevé). En effet, du point de vue de l'expressivité, les résultats que nous avons obtenus lors de la comparaison des TPN et des TA s'étendent très facilement à leurs extensions à chronomètres. Cela signifie que les automates à chronomètres sont strictement plus expressifs que les TPN à chronomètres bornés en terme de bisimulation. Par conséquent, l'indécidabilité de l'accessibilité d'états des automates à chronomètres ne s'étend pas nécessairement aux TPN à chronomètres. Nous avons donc démontré dans [BLRV04, BLRV05] que l'accessibilité d'états pour les TPN à chronomètres peut être réduite au problème de l'arrêt d'une machine à deux compteurs qui est connu indécidable [Min61] ce qui répond définitivement à la question. L'encodage utilisé est apparenté à celui utilisé dans [HKPV95, HKPV98] pour démontrer l'indécidabilité de l'accessibilité pour une

sous-classe d'automates hybrides, mais il est évidemment très différent, notamment parce que les TPN ne manipulent pas les horloges explicitement.

Étant donnée cette indécidabilité, nous avons proposé un semi-algorithme calculant une abstraction de l'espace d'états en termes de classe d'états [LR03a, RL04]. Cette méthode est exacte mais repose sur l'utilisation de polyèdres généraux et est par conséquent très coûteuse en espace et en temps. Nous avons amélioré le semi-algorithme dans [MLR05] en codant les domaines temporels des classes d'état par des DBM dès que cela est possible.

De plus, afin d'obtenir une convergence de l'algorithme et d'améliorer son efficacité, nous avons proposé une surapproximation du polyèdre caractérisant le domaine temporel dans une classe d'états par le plus petit polyèdre représentable par une DBM incluant le polyèdre exact. La méthode est efficace, mais les surapproximations obtenues sont parfois trop grossières. Nous avons donc étudié dans [BLRV04, BLRV05] des surapproximations plus fines qui ont été implémentées dans Tina. Les autres méthodes que nous avons décrites sont implémentées dans l'outil Roméo.

2.3.2 Application à la vérification de système temps réel avec ordonnancement préemptif.

Modéliser un système temps réel avec ordonnancement préemptif par des automates à chronomètres n'est pas toujours simple. L'application est décrite par un produit d'automates et l'ordonnanceur doit être modélisé explicitement. De plus, certaines caractéristiques des systèmes temps réel ne sont pas exprimables avec les automates. L'exemple du sémaphore non borné *a priori* (ou dont on ne connaît pas la borne) illustre ce propos à la fois pour les modèles temporisés ainsi que pour les modèles à chronomètres. Enfin, une modélisation générique d'un système temps réel directement sous la forme d'automates à chronomètres requiert typiquement de faire le produit d'un automate par tâche ainsi qu'un automate par ordonnanceur (donc par processeur) ce qui nous donne au moins autant de chronomètres. Or, la complexité de la vérification d'un automate hybride est exponentielle en le nombre de chronomètres et par conséquent quand ce nombre augmente, le calcul de l'espace d'états est rapidement infaisable.

Les réseaux de Petri temporels étendus à l'ordonnancement (*Scheduling*-TPN) ont été introduits dans [RD02]. Ils étendent les réseaux de Petri temporels en incluant dans la sémantique du modèle le comportement des ordonnanceurs temps réel. Cela implique la suspension et la reprise de l'exécution des tâches, lors des préemptions. Le temps s'arrêtant pour les tâches préemptées, ce modèle s'appuie sur le concept de chronomètre (ou *stopwatch*), horloge pour laquelle le temps peut être arrêté. Pour tout marquage M du réseau, nous définissons un sous-ensemble de M appelé *marquage actif*, et noté $Act(M)$, qui sensibilise exactement les transitions pour lesquelles le temps s'écoule. Ces transitions modélisent soit une tâche qui s'exécute (non préemptée), soit un service de l'exécutif. Dans le premier cas, une transition sensibilisée par M mais pas par $Act(M)$ représente une tâche *prête* mais qui ne s'exécute pas (non active).

La méthode que nous avons proposée dans [LR04, LR05b] consiste en un précalcul qui transforme un *Scheduling*-TPN en automate à chronomètres temporellement bisimilaire. Ce précalcul est conçu de manière à minimiser le nombre de chronomètres de l'automate. De plus, nous pouvons faire ce précalcul en utilisant une surapproximation (utilisant pour chaque classe d'états, le plus petit polyèdre représentable par une DBM incluant le domaine temporel de la classe) et obtenir quand même un automate à chronomètres bisimilaire temporellement

au *Scheduling*-TPN car les localités ajoutées par la surapproximation ne sont pas accessibles dans l'automate résultat.

Ainsi, cette méthode permet, à partir d'un *Scheduling*-TPN, de calculer efficacement un automate à chronomètres temporellement bisimilaire possédant moins de chronomètres qu'une modélisation directe, permettant ainsi d'augmenter de façon importante la taille des systèmes vérifiables en pratique. La méthode est implémentée dans notre outil Roméo qui produit l'automate à chronomètres au format d'entrée de l'outil HyTech.

2.4 Perspectives

Les perpectives immédiates de mes travaux portent principalement sur trois points (d'autres perspectives seront développées dans la conclusion générale de ce document) :

Expressivité. Il existe plusieurs extensions temporelles des réseaux de Petri pour lesquelles le temps peut être associé aux transitions, aux places, aux arcs ou aux jetons. Seulement quelques travaux comparent ces différentes sémantiques et les résultats sont contradictoires. En nous aidant des résultats obtenus sur l'expressivité des réseaux de Petri T-temporels (où le temps est associé aux transitions) et des automates temporisés, nous pouvons certainement contribuer à comparer ces différentes extensions temporelles des réseaux de Petri en terme de langage et de bisimulation et affiner ainsi la classification sur les modèles temporisés.

Contrôle de la non-interférence. Récemment, nous avons posé avec John Mullins et Guillaume Gardey les bases concernant la non-interférence dans le contexte temporisés. Cela ouvre un grand nombre de problèmes, particulièrement sur le contrôle de la non interférence. En effet, les problèmes de décidabilité du contrôle pour plusieurs classes de non interférence sont ouverts. J'ai l'intention de m'y consacrer pour tenter de les résoudre, de trouver les méthodes et algorithmes permettant de décider de l'existence de ces contrôleurs puis de les synthétiser et enfin d'en trouver une implémentation.

Modèles à chronomètres. Les méthodes que nous avons proposées sur les modèles à chronomètres peuvent être généralisées à d'autres modèles et également à d'autres politiques d'ordonnancement (que celles par priorités fixes). De plus nous avons commencé dans le cadre de la thèse de Morgan Magnin à étudier les approches impliquant une sémantique à temps discret qui paraissent prometteuses grâce aux progrès réalisés dans le calcul et la réprésentation symbolique d'espaces d'états de très grande taille pour les réseaux de Petri non temporisés [MDR02, Cia04].

Deuxième partie

Exposés scientifiques détaillés

Chapitre 3

Introduction

Cette partie ne détaillera pas tous les thèmes décrits dans la présentation synthétique de mes travaux (chapitre 2). En effet, seulement trois points seront développés, chacun de ces points illustrant une façon de comparer et de faire coopérer les automates temporisés et les réseaux de Petri temporels ainsi que leurs extensions à chronomètres.

3.1 Le fil conducteur

Les travaux sur les réseaux de Petri temporels et les automates temporisés évoluent au sein de deux communautés de manière souvent indépendante. En effet, en « grossissant le trait », on peut écrire que les réseaux de Petri temporels sont davantage étudiés dans les laboratoires ayant une sensibilité ou une histoire axée sur les thèmes liés à l'automatique alors que les automates temporisés sont davantage étudiés par les laboratoires d'informatique.

Que ce soit une conséquence ou non, très peu d'études comparent ces deux modèles ou proposent des méthodes utilisant les caractéristiques et les avantages des deux modèles. De plus, bien que les travaux sur les abstractions de l'espace d'états des réseaux de Petri temporels soient antérieurs à ceux sur les automates temporisés, aucun outil de model-checking (c'est-à-dire permettant, à partir du modèle d'un système S et d'une propriété φ, de décider $S \models \varphi$) n'était disponible sur les TPN (lorsque nous avons débuté ces travaux) alors que plusieurs méthodes et outils efficaces (Uppaal, Kronos) permettent depuis bientôt 10 ans la vérification de propriétés exprimées avec la logique temporelle TCTL [ACD90] sur les automates temporisés.

Les chapitres à suivre tentent de pallier à ces manques de trois manières :

- en comparant l'expressivité des TPN et des TA en termes d'acceptation de langage temporisé et de bisimulation temporelle ;
- en proposant une logique TCTL pour les TPN (*TPN-TCTL*), en prouvant sa décidabilité et en adaptant aux réseaux de Petri temporels une méthode efficace de model-checking à la volée d'un sous-ensemble de TCTL mise au point sur les automates temporisés dans l'outil Uppaal ;
- en proposant une méthode de vérification d'applications temps réel en présence d'un ordonnancement préemptif utilisant les extensions à chronomètres des réseaux de Petri et des automates par le biais d'une traduction et ce, en prenant avantage de chacun d'eux.

3.2 Plan de cette partie

En plus de ce chapitre, cette partie est composée de 4 chapitres :

- un chapitre de présentation générale des modèles temporisés ;
- un chapitre sur la comparaison de l'expressivité des réseaux de Petri temporels et des automates temporisés en termes d'acceptation de langage et de bisimulation temporelle ;
- un chapitre étudiant le problème de la vérification des propriétés exprimées avec la logique temporelle *Timed Computation Tree Logic* (TCTL) [ACD93] sur les réseaux de Petri temporels ;
- un chapitre sur la vérification d'applications temps réel en présence d'ordonnancement préemptif basée sur l'utilisation de modèle à chronomètres.

Chapitre 4

Modèles temporisés

4.1 Notations générales

- L'ensemble $\mathbb{B}$ désigne les valeurs booléennes `true` et `false` ;
- $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ désignent respectivement les ensembles des entiers naturels, des rationnels et des réels ;
- $\mathbb{R}_{\geq 0}$ désigne l'ensemble des réel positifs ou nuls et $\mathbb{R}_{>0} = \mathbb{R}_{\geq 0} - \{0\}$ est l'ensemble des réels strictement positifs ; il en est de même pour les ensembles $\mathbb{N}$ et $\mathbb{Q}$;
- soit $n \in \mathbb{N}$, $\mathbb{R}^n$ désigne l'espace réel à n dimensions ;
- soit un ensemble fini E. Nous notons $|E|$ le cardinal de E ;
- B^A désigne l'ensemble des applications de A dans B. Si A est fini et $|A| = n$, un élément de B^A est aussi un vecteur dans B^n. Les opérateurs usuels $+, -, <, \leq, >, \geq$ et $=$ sont étendus (élément par élément) aux vecteurs de A^n avec $A = \mathbb{N}, \mathbb{Q}, \mathbb{R}$;
- une *valuation* ν sur un ensemble de variables X est un élément de $\mathbb{R}_{\geq 0}^X$. Pour $\nu \in \mathbb{R}_{\geq 0}^X$ et $d \in \mathbb{R}_{\geq 0}$, $\nu + d$ désigne la valuation $(\nu + d)(x) = \nu(x) + d$, et pour $X' \subseteq X$, $\nu[X' \mapsto 0]$ désigne la valuation ν' avec $\nu'(x) = 0$ si $x \in X'$ et $\nu'(x) = \nu(x)$ sinon. $\overline{0}$ désigne la valuation telle que $\forall x \in X, \nu(x) = 0$;
- soit X un ensemble de variables, une *contrainte atomique* sur X est une formule de la forme $x \sim c$ avec $x \in X$, $c \in \mathbb{Q}_{\geq 0}$ et $\sim \in \{<, \leq, \geq, >\}$. $\mathcal{C}(X)$ désigne l'ensemble des *contraintes* sur l'ensemble de variables X constitué de la conjonction des contraintes atomiques sur X. De plus, nous notons $\mathcal{C}_{dbm}(X)$ l'ensemble des combinaisons booléennes (avec les opérateurs logiques $\vee$, $\wedge$ et $\neg$) de termes de la forme $x - x' \sim c$ ou $x \sim c$, avec $x, x' \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ et $c \in \mathbb{Q}$;
- pour une contrainte $\varphi \in \mathcal{C}(X)$ et une valuation $\nu \in \mathbb{R}_{\geq 0}^X$, nous notons $\varphi(\nu) \in \mathbb{B}$ la valeur de vérité de φ obtenue en substituant chaque occurrence de x dans φ par $\nu(x)$. Nous notons ainsi $[\![\varphi]\!] = \{\nu \in \mathbb{R}_{\geq 0}^X \mid \varphi(\nu) = \texttt{true}\}$.

4.2 Langages et systèmes de transitions temporisés

4.2.1 Mots et langages temporisés

Soit Σ un ensemble (ou alphabet). Σ^* (respectivement Σ^ω) est l'ensemble des suites finies (respectivement infinies) d'éléments de Σ et $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Nous parlerons de mots finis sur l'alphabet Σ pour les éléments de Σ^* et de mots infinis sur l'alphabet Σ pour les éléments de

Σ^ω.

Nous notons $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ avec $\varepsilon \notin \Sigma$, où ε représente une action (lettre) particulière dite *silencieuse* ou *non observable*.

Définition 1 (Mots temporisés) *Un* mot temporisé w *sur un alphabet* Σ *est une séquence finie ou infinie* $w = (a_0, d_0)(a_1, d_1)\cdots(a_n, d_n)\cdots$ *telle que pour tout* $i \geq 0$, $a_i \in \Sigma$, $d_i \in \mathbb{R}_{\geq 0}$ *et* $d_{i+1} \geq d_i$.

Un mot temporisé $w = (a_0, d_0)(a_1, d_1)\cdots(a_n, d_n)\cdots$ sur Σ est une paire $(a, d) \in \Sigma^\infty \times \mathbb{R}^\infty_{\geq 0}$ avec $|a| = |d|$. La valeur d_i donne la date absolue (en considérant que l'instant initial est la date 0) de l'action a_i.

Nous notons $\mathit{Untimed}(w) = a_0 a_1 \cdots a_n \cdots$ pour la partie non temporisé de w, et $\mathit{Duration}(w) = \sup_{d_i \in d} d_i$ pour la durée de w.

Définition 2 (Langages temporisés) *Notons* $\mathcal{TW}^*(\Sigma)$ *(respectivement* $\mathcal{TW}^\omega(\Sigma)$*) l'ensemble des mots temporisés finis (respectivement infinis) sur* Σ *et* $\mathcal{TW}^\infty(\Sigma) = \mathcal{TW}^*(\Sigma) \cup \mathcal{TW}^\omega(\Sigma)$. *Un* langage temporisé L *sur* Σ *est un sous-ensemble de* $\mathcal{TW}^\infty(\Sigma)$.

4.2.2 Systèmes de transitions temporisés

Nous nous intéressons à des systèmes pouvant être décrits par un système de transitions, c'est-à-dire un ensemble (quelconque) d'états et de transitions étiquetées par des actions entre les états. Lorsque l'on se trouve dans l'un des états d'un système de transitions, il est possible de changer d'état en effectuant une action qui étiquette l'une des transitions sortant de l'état. Une exécution dans un système de transitions est alors une séquence d'actions qui est souvent notée sous la forme d'un mot représentant la succession des actions.

Les systèmes de transitions temporisées (*Timed Transition System* en anglais, ou TTS) sont des systèmes de transitions particuliers pour lesquels deux types de transitions sont possibles : des transitions d'action et des transitions de temps modélisant respectivement des évolutions *discrètes* et des évolutions *continues* du système.

Définition 3 (Système de transitions temporisé) *Un* système de transitions temporisé (TTS) *sur un ensemble d'action* Σ *(ou* alphabet*) est un quadruplet* $S = (Q, Q_0, \Sigma, \longrightarrow)$ *où* Q *est un ensemble d'état,* $Q_0 \subseteq Q$ *est un ensemble d'états initiaux,* Σ *est un ensemble fini d'actions (disjoint de* $\mathbb{R}_{\geq 0}$*),* $\longrightarrow \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ *est une relation de transition (ensemble d'arcs). Si* $(q, e, q') \in \longrightarrow$*, nous notons aussi* $q \xrightarrow{e} q'$*. La relation de transition se décompose en une relation de transition* continue $\xrightarrow{d \in \mathbb{R}_{\geq 0}}$ *et une relation de transition* discrète $\xrightarrow{a \in A}$.

Nous faisons les hypothèses habituelles suivantes sur les TTS :
- 0-délai : $q \xrightarrow{0} q'$ ssi $q = q'$,
- Additivité : si $q \xrightarrow{d} q'$ et $q' \xrightarrow{d'} q''$ avec $d,\ d' \in \mathbb{R}_{\geq 0}$, alors $q \xrightarrow{d+d'} q''$,
- Continuité : si $q \xrightarrow{d} q'$, alors pour tout d' et d'' dans $\mathbb{R}_{\geq 0}$ tel que $d = d' + d''$, il existe q'' tel que $q \xrightarrow{d'} q'' \xrightarrow{d''} q'$,
- Déterminisme temporel : si $q \xrightarrow{d} q'$ et $q \xrightarrow{d} q''$ avec $d \in \mathbb{R}_{\geq 0}$, alors $q' = q''$.

Définition 4 (Exécution (run) d'un TTS) *Une* exécution ρ *d'un TTS* S *est une séquence finie ou infinie de transitions continues et discrètes de* S. *L'ensemble des exécutions d'un TTS* S *est noté* $[\![S]\!]$.

Une exécution peut toujours s'écrire sous la forme d'une alternance de transition continue (éventuellement de durée 0) et de transition discrète :

$$\rho = q_0 \xrightarrow{d_0} q'_0 \xrightarrow{a_0} q_1 \xrightarrow{d_1} q'_1 \xrightarrow{a_1} \cdots q_n \xrightarrow{d_n} q'_n \cdots$$

Pour une exécution de taille n, nous notons $\mathit{Untimed}(\rho) = a_0 a_1 \cdots$ et $\mathit{Duration}(\rho) = \sum_{i=0}^{n} d_i$.

Définition 5 (Trace) *La* trace *d'une exécution* $\rho = q_0 \xrightarrow{d_0} q'_0 \xrightarrow{a_0} q_1 \cdots q_k \xrightarrow{d_k} q'_k \cdots$ *d'un TTS est le mot temporisé* $\mathit{trace}(\rho) = (a_0, \delta_0) \cdots (a_k, \delta_k) \cdots$ *avec* $\delta_k = \sum_{i=0}^{k} d_i$.

Un mot temporisé $w = (a_i, d_i)_{0 \leq i \leq n}$ est *accepté* par le système de transition $S = (Q, Q_0, \Sigma, \longrightarrow)$ si il existe une exécution de S à partir d'un état initial $q_0 \in Q_0$ et de trace w. Le *langage temporisé* $\mathcal{L}(S)$ accepté par S est l'ensemble des mots temporisés acceptés par S.

Le *langage non temporisé* de S est constitué des mots de S dans lesquels les actions continues ont été abstraites.

Définition 6 (TTS ε-abstrait) *Soit* $S = (Q, Q_0, \Sigma_\varepsilon, \longrightarrow)$ *un TTS. Nous définissons le TTS* $S^\varepsilon = (Q, Q_0^\varepsilon, \Sigma, \longrightarrow_\varepsilon)$ *dans lequel les actions* ε *ont été abstraites de* S *par :*

- $q \xrightarrow{d}_\varepsilon q'$ *avec* $d \geq \mathbb{R}_{\geq 0}$ *ssi il existe une exécution* $\rho = q \to q'$ *avec* $\mathit{Untimed}(\rho) = \varepsilon^*$ *et* $\mathit{Duration}(\rho) = d$,
- $q \xrightarrow{a}_\varepsilon q'$ *avec* $a \in \Sigma$ *ssi il existe une exécution* $\rho = q \to q'$ *avec* $\mathit{Untimed}(\rho) = \varepsilon^* a \varepsilon^*$ *and* $\mathit{Duration}(\rho) = 0$,
- $Q_0^\varepsilon = \{q \mid \exists q' \in Q_0 \mid \ q' \to q \text{ et } \mathit{Duration}(\rho) = 0 \wedge \mathit{Untimed}(\rho) = \varepsilon\}$.

Enfin si S est défini sur Σ_ε, le langage accepté par S est constitué des mots dans lesquels les actions ε ont été abstraites.

4.3 Bisimulation et expressivité de modèles temporisés

4.3.1 Simulation et bisimulation temporelles

Définition 7 (Simulation temporelle forte) *Soient deux systèmes de transitions temporisés* $S_1 = (Q_1, Q_0^1, \Sigma, \longrightarrow_1)$ *et* $S_2 = (Q_2, Q_0^2, \Sigma, \longrightarrow_2)$ *sur* Σ *et* $\sqsubseteq$ *une relation binaire sur* $Q_1 \times Q_2$. *Nous écrivons* $s \sqsubseteq s'$ *pour* $(s, s') \in \sqsubseteq$. $\sqsubseteq$ *est une* relation de simulation temporelle forte *de* S_1 *par* S_2 *si les trois assertions suivantes sont vérifiées :*

1. *si* $s_1 \in Q_0^1$, *alors il existe* $s_2 \in Q_0^2$ *tel que* $s_1 \sqsubseteq s_2$;
2. *si* $s_1 \xrightarrow{d}_1 s'_1$ *avec* $d \in \mathbb{R}_{\geq 0}$ *et* $s_1 \sqsubseteq s_2$ *alors il existe* $s_2 \xrightarrow{d}_2 s'_2$ *tel que* $s'_1 \sqsubseteq s'_2$;
3. *si* $s_1 \xrightarrow{a}_1 s'_1$ *avec* $a \in \Sigma$ *et* $s_1 \sqsubseteq s_2$ *alors il existe* $s_2 \xrightarrow{a}_2 s'_2$ *tel que* $s'_1 \sqsubseteq s'_2$.

Un TTS S_2 simule fortement S_1 *si il existe une relation de simulation forte de* S_1 *par* S_2. *Nous notons alors* $S_1 \sqsubseteq_{\mathcal{S}} S_2$.

Définition 8 (Simulation temporelle faible) *Soient deux systèmes de transitions temporisés* $S_1 = (Q_1, Q_0^1, \Sigma_\varepsilon, \longrightarrow_1)$ *et* $S_2 = (Q_2, Q_0^2, \Sigma_\varepsilon, \longrightarrow_2)$ *sur* Σ_ε *et* $\sqsubseteq$ *une relation binaire sur* $Q_1 \times Q_2$. $\sqsubseteq$ *est une* relation de simulation temporelle faible *de* S_1 *par* S_2 *si c'est une relation de simulation temporelle forte entre ces deux TTS* ε*-abstraits. Un TTS* S_2 simule faiblement S_1 *si il existe une relation de simulation faible de* S_1 *par* S_2. *Nous notons alors* $S_1 \sqsubseteq_{\mathcal{W}} S_2$.

Définition 9 (Bisimulation temporelle) *Deux TTS S_1 et S_2 sont en relation de bisimulation temporelle forte (respectivement faible) si il existe une relation de simulation forte (respectivement faible) $\sqsubseteq$ de S_1 par S_2 et si $\sqsubseteq^{-1}$ est aussi une relation de simulation forte*[1] *(respectivement faible) de S_2 par S_1. Nous notons alors $S_1 \approx_{\mathcal{S}} S_2$ (respectivement $S_1 \approx_{\mathcal{W}} S_2$).*

4.3.2 Expressivité des modèles temporisés

Soient $\mathcal{C}$ et $\mathcal{C}'$ deux classes de modèles temporisés.

Définition 10 (Expressivité en termes d'acceptation de langage temporisé) *La classe $\mathcal{C}$ est* plus expressive *que la classe $\mathcal{C}'$ en termes d'acceptation de langage temporisé si pour tout $B' \in \mathcal{C}'$ il existe $B \in \mathcal{C}$ tel que $\mathcal{L}(B) = \mathcal{L}(B')$. Nous notons alors $\mathcal{C}' \leq_{\mathcal{L}} \mathcal{C}$. De plus si il existe $B \in \mathcal{C}$ tel qu'il n'existe pas $B' \in \mathcal{C}'$ avec $\mathcal{L}(B) = \mathcal{L}(B')$, alors $\mathcal{C}' <_{\mathcal{L}} \mathcal{C}$ (« strictement moins expressive »). Si on a $\mathcal{C}' \leq_{\mathcal{L}} \mathcal{C}$ et $\mathcal{C} \leq_{\mathcal{L}} \mathcal{C}'$ alors $\mathcal{C}$ et $\mathcal{C}'$ sont d'expressivité égale en termes d'acceptation de langage temporisé, et nous notons $\mathcal{C} =_{\mathcal{L}} \mathcal{C}'$.*

Définition 11 (Expressivité en termes de bisimulation temporelle) *La classe $\mathcal{C}$ est* plus expressive *que la classe $\mathcal{C}'$ en termes de bisimulation temporelle forte (respectivement faible) si pour tout $B' \in \mathcal{C}'$ il existe $B \in \mathcal{C}$ tel que $B \approx_{\mathcal{S}} B'$ (respectivement $B \approx_{\mathcal{W}} B'$). Nous notons alors $\mathcal{C}' \precsim_{\mathcal{S}} \mathcal{C}$ (respectivement $\mathcal{C}' \precsim_{\mathcal{W}} \mathcal{C}$). De plus, si il existe $B \in \mathcal{C}$ tel qu'il n'existe pas $B' \in \mathcal{C}'$ avec $B \approx_{\mathcal{S}} B'$ (respectivement $B \approx_{\mathcal{W}} B'$), alors $\mathcal{C}' \precnsim_{\mathcal{S}} \mathcal{C}$ (respectivement $\mathcal{C}' \precnsim_{\mathcal{W}} \mathcal{C}$). Si on a $\mathcal{C}' \precsim_{\mathcal{S}} \mathcal{C}$ et $\mathcal{C} \precsim_{\mathcal{S}} \mathcal{C}'$ (respectivement $\precsim_{\mathcal{W}}$) alors $\mathcal{C}$ et $\mathcal{C}'$ sont d'expressivité égale en termes de bisimulation temporelle forte (respectivement faible) et nous notons $\mathcal{C} \approx_{\mathcal{S}} \mathcal{C}'$ (respectivement $\mathcal{C} \approx_{\mathcal{W}} \mathcal{C}'$).*

4.4 Réseaux de Petri T-temporels

Les systèmes de transitions sont très généraux et donc de très (trop) bas niveau. D'une manière générale, il n'est pas possible de représenter de tels systèmes de manière finie. Ils ne sont par conséquent pas directement utilisables pour la vérification ou le contrôle. Nous nous intéressons donc à des modèles que l'on peut représenter de manière finie mais dont la sémantique sera donnée par un système de transitions.

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

4.4.1 Présentation informelle

Les réseaux de Petri T-temporels [Mer74], ou *time Petri nets* (TPN) en anglais, étendent les réseaux de Petri avec des intervalles $[\alpha(t), \beta(t)]$ associés à chaque transition t du réseau Pour être tirée, une transition t doit non seulement être sensibilisée mais également l'avoir été continûment pendant une durée comprise entre $\alpha(t)$ et $\beta(t)$.

[1] $S_2 \sqsubseteq^{-1} S_1$ ssi $S_1 \sqsubseteq S_2$.

4.4.2 Définitions

Définition 12 (Réseau de Petri T-temporel) *Un réseau de Petri T-temporel est un 7-uplet* $\mathcal{N} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0)$*, où*

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

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

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

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

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

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

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

Nous définissons la sémantique des réseaux de Petri T-temporels sous la forme d'un système de transitions temporisé (TTS).

Définition 13 (Sémantique d'un TPN) *La sémantique d'un réseau de Petri T-temporel* $\mathcal{N}$ *est définie sous la forme d'un système de transitions temporisé* $\mathcal{S}_\mathcal{N} = (Q, q_0, \Sigma, \rightarrow)$ *tel que :*

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

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

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

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

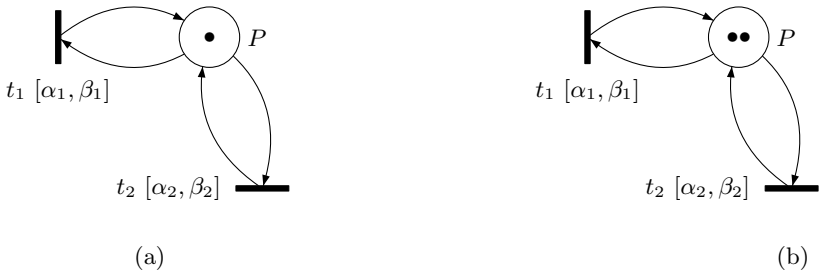


Fig. 4.1 – Example de transitions nouvellement sensibilisées

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

Définition 14 (Transition tirable) *Soit $s = (M, \nu)$ un état de la sémantique d'un réseau de Petri T-temporel. Une transition t est dite* tirable *dans s si $M \geq {}^\bullet t$ et $\alpha(t) \leq \nu(t) \leq \beta(t)$.*

Nous pouvons remarquer que, dans cette sémantique, si une place contient plusieurs jetons pouvant sensibiliser une ou des transitions sortantes, seul le nombre de jetons indiqué sur l'arc sera considéré. Les autres seront utilisés pour les tirs suivants éventuels des transitions sortantes. C'est une hypothèse *monoserveur*. Cette sémantique est illustrée par la figure 4.1 : supposons que t_1 est tirable. Sur la figure 4.1a, t_1 et t_2 sont sensibilisées par le marquage M et par le marquage $M' = M - {}^\bullet t_1 + t_1{}^\bullet$ mais pas par $M - {}^\bullet t_1$. Les transitions t_1 et t_2 sont donc *nouvellement* sensibilisées par le tir de t_1.

Sur la figure 4.1b, t_1 et t_2 sont sensibilisées par le marquage M et par le marquage $M' = M - {}^\bullet t_1 + t_1{}^\bullet$, mais aussi par $M - {}^\bullet t_1$. Dans ce cas, t_1 est *nouvellement* sensibilisée par le tir de t_1 (car elle est la transition tirée) mais pas t_2 : t_2 *reste* sensibilisée.

On peut complexifier la sémantique de façon à ce qu'une transition puisse être sensibilisée plusieurs fois simultanément [Ber01]. On considère alors une hypothèse *multiserveur* et le nombre d'horloges du réseau devient potentiellement infini.

Définition 15 (Réseau de Petri temporel étiqueté) *Un réseau Petri temporel étiqueté (*Labelled Time Petri Net*) $\mathcal{N}$ est un 9-uplet $\mathcal{N} = (P, T, \Sigma_\varepsilon, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0, \Lambda)$ tel que : $(P, T, {}^\bullet(.), (.)^\bullet, , \alpha, \beta, M_0)$ est un TPN, Σ est un ensemble fini d'*actions *et $\Lambda : T \to \Sigma_\varepsilon$ est une fonction de nommage. Un TPN non étiqueté est donc un TPN étiqueté tel que $\Sigma = T$ et $\Lambda(t) = t$ pour tout $t \in T$.*

Classiquement (et c'est la convention que nous utilisons dans ce manuscrit), par TPN, nous désignons implicitement des TPN labélisés avec $\Sigma = T$ et $\Lambda(t) = t$.

Enfin nous pouvons étendre très naturellement les définitions 12, 13 et 15 pour considérer aussi des contraintes strictes dans les intervalles de tir des transitions.

4.4.3 Problèmes classiques

Étant donné un réseau de Petri T-temporel $\mathcal{N}$ et $\mathcal{S}_\mathcal{N}$ sa sémantique, plusieurs problèmes peuvent être étudiés, notamment :

– l'*accessibilité de marquages* : étant donné un marquage M, « $\exists (M', \nu') \in \mathcal{S}_\mathcal{N}, M = M'$ » ;
– la *bornitude* : « $\exists b \in \mathbb{N}, \forall (M, \nu) \in \mathcal{S}_\mathcal{N}, \forall p \in P, M(p) \leq b$ » ;

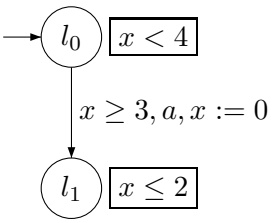


FIG. 4.2 – Un automate temporisé

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

L'accessibilité de marquages est un problème indécidable [JLL77], ce qui implique que la bornitude, l'accessibilité d'états et la vivacité sont également des problèmes indécidables. En revanche, la k-bornitude est décidable et peut être décidée par le calcul d'une abstraction finie de l'espace d'états telle que le graphe des classes d'états de [BD91].

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

4.5 Automates temporisés

4.5.1 Présentation

Les automates temporisés, ou *timed automata* (TA) en anglais, ont été introduits par Alur et Dill en 1994 [AD94] et étendus avec la notion d'invariant par Henzinger *et al.* dans [HNSY94]. Le temps est ajouté au modèle classique des automates sous la forme d'horloges et de prédicats sur ces horloges. Ces prédicats sont de deux types : les *gardes*, associées aux transitions discrètes, donnent des contraintes sur les horloges à respecter pour pouvoir exécuter cette transition discrète. Les *invariants*, associés aux localités, donnent des contraintes qui doivent être respectées dans les localités.

La figure 4.2 donne un exemple très simple d'automate temporisé. La localité initiale est l_0. L'automate dispose d'une seule horloge : x. x est nul à l'instant initial et l'invariant de l_0 indique donc que l'on pourra rester dans l_0 strictement moins de 4 unités de temps. Dès que 3 unités de temps se sont écoulées, la garde de la transition entre l_0 et l_1 est vérifiée (et l'invariant de l_1 est vérifié après franchissement de cette transition) et la transition peut donc être franchie. Si cette transition est franchie, l'horloge x est remise à zéro et l_1 devient la localité active.

4.5.2 Définitions

Définition 16 (Automate temporisé) *[HNSY94] Un* automate temporisé *est un 6-uplet* $(L, l_0, X, \Sigma, E, Inv)$ *où*

– L *est un ensemble fini de* localités *;*
– l_0 *est la* localité initiale *;*
– X *est un ensemble fini d'*horloges *à valeurs réelles positives ;*

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

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

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

Définition 17 (Sémantique d'un automate temporisé) *La sémantique d'un automate temporisé* $\mathcal{A}$ *est définie sous la forme d'un système de transitions temporisé* $\mathcal{S}_{\mathcal{A}} = (Q, Q_0, \Sigma, \rightarrow)$ *où*

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

Il est courant de décrire un système comme une composition parallèle d'automates temporisés. Dans ce but, on utilise la notion classique de composition basée sur une *fonction de synchronisation* à la Arnold-Nivat [Arn94]. Soient $\mathcal{A}_1, \ldots, \mathcal{A}_n$ n automates temporisés avec $\mathcal{A}_i = (N_i, l_{i,0}, C_i, \Sigma, E_i, Inv_i)$. Une *fonction de synchronisation* f est une fonction partielle de $(\Sigma \cup \{\bullet\})^n \hookrightarrow \Sigma$ pour laquelle $\bullet$ est un symbole qui indique qu'un automate n'est pas impliqué dans un pas du système global. Nous notons $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$ la composition parallèle des $\mathcal{A}_i$ synchronisés par f. Les états de $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$ sont des paires $(\overline{l}, v)$ avec $\overline{l} = (l_1, \ldots, l_n) \in N_1 \times \ldots \times N_n$ et $v = v_1 \cdots v_n$ avec[2] $v_i \in (\mathbb{R}_{\geq 0})^{C_i}$ (nous supposons que les ensembles d'horloges C_i sont disjoints). La sémantique d'un produit synchronisé d'automates temporisés est un système de transitions temporisé : le produit synchronisé peut évoluer par une transition discrète ou par l'écoulement du temps si tous les composants du produit l'autorisent. Cela est formalisé par la définition suivante :

Définition 18 (Sémantique d'un produit d'automates temporisés) *Soient* $\mathcal{A}_1, \ldots, \mathcal{A}_n$*, n automates temporisés avec* $\mathcal{A}_i = (N_i, l_{i,0}, C_i, \Sigma, E_i, Inv_i)$*, et* f *une fonction de synchronisation partielle de* $(\Sigma \cup \{\bullet\})^n \hookrightarrow \Sigma$*. La sémantique de* $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$ *est un système de transitions temporisé* $S = (Q, q_0, \rightarrow)$ *avec* $Q = N_1 \times \ldots \times N_n \times (\mathbb{R}_{\geq 0})^C$*,* q_0 *est l'état initial* $((l_{1,0}, \ldots, l_{n,0}), \overline{0})$ *et* $\rightarrow$ *est définie par :*

- $(\overline{l}, v) \xrightarrow{b} (\overline{l'}, v')$ *ssi il existe* $(a_1, \ldots, a_n) \in (\Sigma \cup \{\bullet\})^n$ *tel que* $f(a_1, \ldots, a_n) = b$ *et pour tout* i *nous avons :*
  - *. si* $a_i = \bullet$*, alors* $l'_i = l_i$ *et* $v'_i = v_i$*,*
  - *. si* $a_i \in \Sigma$*, alors* $\xrightarrow{(l_i, v_i)} a_i(l'_i, v'_i)$*.*

[2] v_i est la restriction de v sur C_i.

– $(\bar{l}, v) \xrightarrow{t} (\bar{l}, v')$ *ssi* $\forall i \in [1..n]$, *nous avons* $\overset{(l_i, v_i)}{\rightarrow} t(l_i, v'_i)$

Nous pouvons, à partir du produit de n automates temporisés, construire (syntaxiquement) un nouvel automate temporisé dont la sémantique est celle décrite ci-dessus [LPY95].

4.6 Classes de TPN et de TA

Nous notons $\mathcal{TA}$ la classe des TA (automates temporisés) et $\mathcal{TPN}$ la classe des TPN (réseaux de Petri temporels).

Définition 19 (Classes de réseaux de Petri temporels bornés) *Soit k-$\mathcal{TPN}$, l'ensemble des TPN k-bornés.*
Soit 1-$\mathcal{TPN}$, l'ensemble des TPN 1-bornés c'est-à-dire saufs.
Soit B-$\mathcal{TPN} = \{\mathcal{N} | \exists k \geq 0 \, | \mathcal{N} \in k\text{-}\mathcal{TPN}\}$, l'ensemble des TPN bornés.

Définition 20 (Classe sur contraintes temporelles larges ou strictes) *Soit $\mathcal{C}$ une classe de modèle temporisé. Par défaut cette classe est définie sur des contraintes larges et strictes (contraintes temporelles définies avec les opérateurs $<, \leq, >$ et $\geq$). La restriction de cette classe aux contraintes larges est notée $\mathcal{C}(\leq, \geq)$.*

Ainsi :
- la classe des TA de [HNSY94] avec contraintes strictes et contraintes larges est notée $\mathcal{TA}$;
- la classe des TA sans contrainte stricte est notée $\mathcal{TA}(\leq, \geq)$;
- la classe des TPN « à la Merlin » c'est-à-dire sans contrainte stricte (donc avec des intervalles fermés (sauf sur ∞)) est notée $\mathcal{TPN}(\leq, \geq)$;
- la classe des TPN avec des intervalles ouverts et fermés est notée $\mathcal{TPN}$.

Chapitre 5

Expressivité des TPN par rapport aux TA

But de l'étude. Très peu d'études comparent l'expressivité des réseaux de Petri temporels et des automates temporisés. Les réseaux de Petri temporels généraux ont le pouvoir d'expression d'une machine de Turing [JLL77] ce qui n'est pas le cas des automates temporisés. Cela s'exprime en terme de décidabilité par le fait que l'accessibilité est décidable pour les TA ainsi que pour les TPN bornés et ne l'est pas pour les TPN généraux. Nous pouvons alors supposer que les TPN sont strictement plus expressifs que les TA.

C'est en fait plus compliqué que cela et le but de notre étude est de comparer précisément l'expressivité des TA et des TPN en termes d'acceptation de langage temporisé et de bisimulation temporelle.

Nous prouvons que les TPN et les TA ont la même expressivité en termes d'acceptation de langage temporisé. Du point de vue de l'expressivité en terme de bisimulation, nous prouvons que les TA sont strictement plus expressifs que les TPN bornés.

Pour cela nous prouvons d'abord que tout TPN peut être traduit en un TA avec variables qui lui est temporellement bisimilaire. Nous prouvons ensuite qu'il existe un TA dont aucun TPN n'est temporellement bisimilaire (même faiblement). Nous proposons alors une traduction des TA vers les TPN préservant le langage temporisé puis nous identifions la sous-classe syntaxique des TA, équivalente (par bisimulation) aux TPN à contraintes larges.

Contexte des travaux. Nous avons débuté ces travaux avec Franck Cassez en étudiant une traduction structurelle des TPN vers les TA préservant la sémantique comportementale (bisimulation temporelle). Ces travaux se sont poursuivis en 2004 par une collaboration avec Béatrice Bérard et Serge Haddad (du LAMSADE) ainsi qu'avec Didier Lime (en post-doc au CISS à Aalborg pendant cette période) sur l'étude de traductions des TA vers les TPN. Ces travaux ont conduit aux publications suivantes [CR03] [CR04] [BCH+05b] [BCH+05a] [BCH+05c].

5.1 Introduction

Les relations entre TPN et TA n'ont pas été beaucoup étudiées. Dans [SY96], J. Sifakis et S. Yovine se sont principalement intéressés au problème de la *composition*. Ils montrent que pour une sous-classe des réseaux de Petri à flux temporels *saufs*, la notion de composition

utilisée sur les TA n'est pas appropriée pour définir ce type de réseaux de Petri à partir d'une composition de TA. Ils proposent donc des *automates temporisés à échéance* et une notion plus flexible de composition. Dans [BST98], Bornot, Sifakis et Tripakis considèrent des réseaux de Petri à échéances (PND) qui sont des réseaux de Petri saufs étendus avec des horloges. Cela implique que les PND sont équivalents à des automates temporisés à échéances dont la structure discrète est le graphe des marquages du réseau de Petri sous-jacent. Ils proposent une traduction des TPN vers les PND ce qui définit par transitivité une traduction des TPN vers les TA, ceci pour des réseaux saufs.

Dans [CEP00], les auteurs considèrent une extension des TPN (*PRES+*) et une traduction vers les automates hybrides. La correction de la traduction n'est pas prouvée et la méthode n'est définie que pour des réseaux saufs

Sava et Alla [SA01, Sav01] considèrent des TPN bornés dont le réseau sous-jacent n'est pas nécessairement sauf et proposent un algorithme calculant le graphe des marquages d'un TPN sous la forme d'un automate temporisé (avec une horloge par transition du TPN). Cependant, ils ne prouvent pas que l'automate obtenu est *bisimilaire* au réseau initial ou équivalent dans un sens formellement défini.

Nous avons proposé dans [LR03b, LR05a] une extension de la construction du graphe des classes d'états qui permet de construire, pour un TPN borné, ce graphe sous la forme d'un automate temporisé. Le TA obtenu et le TPN initial sont temporellement bisimilaires et, relativement à un critère donné, le nombre d'horloge obtenu dans le TA est minimal.

Pour les deux dernières méthodes que nous venons de décrire, les traductions proposées sont limitées à des réseaux bornés et nécessitent le calcul de l'espace d'états du TPN. Les méthodes précédentes en revanche sont structurelles mais sont limitées à des TPN saufs (ou même dont le réseau sous-jacent est sauf).

Concernant la traduction inverse (c'est à dire des TA vers les TPN) les travaux sont encore plus rares. Dans [HKSLT02], les auteurs comparent les *Timed State Machines* (TSM) et les TPN. Ils proposent une traduction des TSM vers les TPN qui préservent le langage temporisé. La sémantique considérée est une sémantique faible et les auteurs ne considèrent que des contraintes temporelles avec des intervalles fermés. Dans [BD99], les auteurs considèrent le problème de l'expressivité mais seulement pour une sous-classe des TPN et concluent que les TPN saufs avec contraintes larges sont strictement moins expressifs que les TA avec contraintes quelconques (strictes et larges).

5.2 Traduction des TPN vers les TA préservant la bisimulation

Dans cette section, à partir d'un TPN, nous construisons structurellement (syntaxiquement) un produit synchronisé d'automates temporisés dont les comportements sont temporellement bisimilaires à ceux du TPN intial.

5.2.1 Traduction structurelle des TPN vers les TA

Soit un réseau de Petri temporel $\mathcal{N} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ avec $P = \{p_1, \cdots, p_m\}$ et $T = \{t_1, \cdots, t_n\}$.

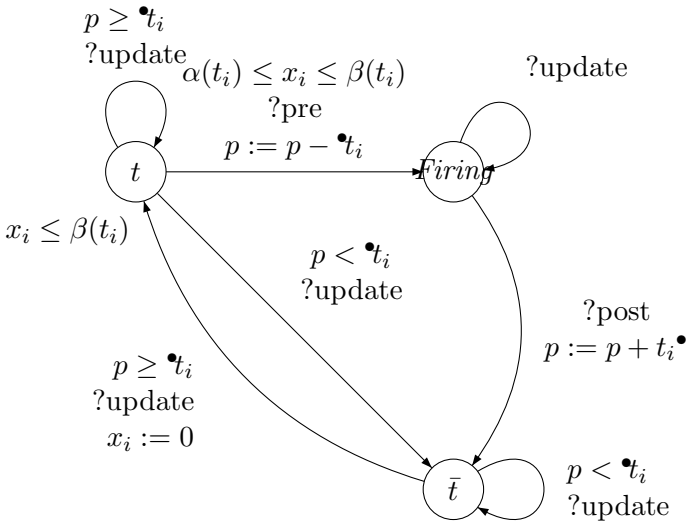


FIG. 5.1 – Automate $\mathcal{A}_i$ correspondant à la transition t_i

Automate temporisé associé à une transition du TPN. Nous définissons un automate temporisé $\mathcal{A}_i$ pour chaque transition t_i de T (voir figure 5.1). Cet automate temporisé possède une horloge x_i. Les états de l'automate $\mathcal{A}_i$ donnent l'état de la transition t_i : dans l'état t, la transition est sensibilisée ; dans l'état $\bar{t}$, elle n'est pas sensibilisée ; dans l'état *Firing*, elle est en train d'être tirée. L'état initial de chaque $\mathcal{A}_i$ dépend du marquage initial M_0 du TPN qui est traduit. Si $M_0 \geq {}^\bullet t_i$ alors l'état initial est t et, dans le cas contraire, c'est $\bar{t}$. Cet automate met à jour un tableau d'entiers p qui est le vecteur de marquage et qui est partagé par tous les $\mathcal{A}_i$'s. Ceci n'est pas couvert par la définition 18 mais celle-ci est très souvent étendue ([LPY97, PL00]) avec des variables entières (ce qui ne change pas l'expressivité du modèle si ces variables sont bornées).

Le superviseur. Le superviseur *SU* est décrit figure 5.2. Les localités 1 à 3 indicées par un c sont supposées *urgentes* ou *committed*[1] ce qui signifie que le temps ne peut pas s'écouler lorsque l'on est dans ces localités. L'état initial du superviseur est 0.

Nous définissons la fonction de synchronisation f à $n+1$ paramètres (n est le nombre de transitions du TPN à traduire) par :

- $f(!pre, \bullet, \cdots, ?pre, \bullet, \cdots) = pre_i$ si $?pre$ est le $(i+1)$ème argument et tous les autres arguments sont $\bullet$,
- $f(!post, \bullet, \cdots, ?post, \bullet, \cdots) = post_i$ si $?post$ est le $(i+1)$ème argument et tous les autres arguments sont $\bullet$,
- $f(!update, ?update, \cdots, ?update) = update$.

Nous notons $\Delta(\mathcal{N}) = (SU|\mathcal{A}_1| \cdots \times \mathcal{A}_n)_f$ l'automate temporisé associé au TPN $\mathcal{N}$. Les actions *pre*, *post* et *update* sont des actions ε. Nous allons prouver dans la section suivante que la sémantique de $\Delta(\mathcal{N})$ est *équivalente* à la sémantique de $\mathcal{N}$. Pour cela nous devons associer les états de $\mathcal{N}$ aux états de $\Delta(\mathcal{N})$ et nous définissons l'équivalence suivante :

[1]Dans *SU*, les localités *committed* ou *urgentes* sont équivalentes ; ce type de localité peut être simulé par l'ajout d'une horloge x qui est mise à 0 en entrant dans la localité et en ajoutant l'invariant $x = 0$ à la localité ; pour notre problème, il suffit de considérer que, dans une localité urgente, le temps ne peut s'écouler.

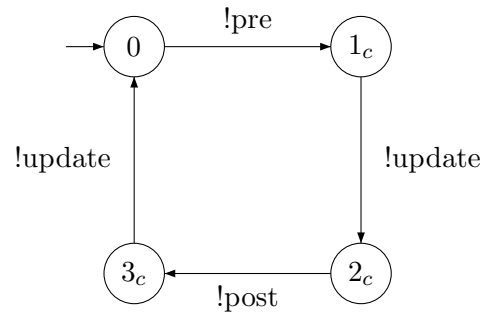


Fig. 5.2 – Automate du superviseur SU

Définition 21 (Équivalence entre états) *Soient (M,ν) un état de $\mathcal{S}_\mathcal{N}$ et $((s,p),\overline{q},v)$ une configuration*[2] *de $\mathcal{S}_{\Delta(\mathcal{N})}$. Nous avons*

$$(M,\nu) \approx ((s,p),\overline{q},v) \text{ ssi } \begin{cases} s = 0, \\ \forall i \in [1..m], \quad p[i] = M(p_i), \\ \forall k \in [1..n], \quad q_k = \begin{cases} t \text{ si } M \geq {}^\bullet t_k, \\ \bar{t} \text{ sinon} \end{cases} \\ \forall k \in [1..n], \quad \nu_k = v_k \end{cases} \qquad \square$$

5.2.2 Correction de la traduction

Nous prouvons maintenant que notre traduction préserve le comportement du TPN initial dans le sens où la sémantique du TPN et sa traduction sont temporellement bisimilaires. Soient un TPN $\mathcal{N}$ et $\mathcal{S}_\mathcal{N} = (Q, q_0, \Sigma, \rightarrow)$ sa sémantique. Soient $\mathcal{A}_i$ l'automate associé à la transition t_i de $\mathcal{N}$ comme décrit par la figure 5.1, SU l'automate superviseur de la figure 5.2 et f la fonction de synchronisation définie précédemment. La sémantique de $\Delta(\mathcal{N}) = (SU|\mathcal{A}_1|\cdots|\mathcal{A}_n)_f$ est le système de transitions temporisé $\mathcal{S}_{\Delta(\mathcal{N})} = (Q_{\Delta(\mathcal{T})}, q_0^{\Delta(\mathcal{N})}, \Sigma_\varepsilon, \rightarrow)$ (les actions *pre*, *post* et *update* sont des actions ε).

Théorème 1 (Bisimilarité temporelle) *Pour tout état (M,ν) de $\mathcal{S}_\mathcal{N}$ et $((0,p),\bar{q},v)$ de $\mathcal{S}_{\Delta(\mathcal{N})}$ tel que $(M,\nu) \approx ((0,p),\bar{q},v)$ nous avons :*

$$(M,\nu) \xrightarrow{t_i} (M',\nu') \quad ssi \quad \begin{cases} ((0,p),\bar{q},v) \overset{w_i}{\Longrightarrow} ((0,p'),\bar{q}',v') \text{ avec} \\ w_i = pre_i.update.post_i.update \text{ et} \\ (M',\nu') \approx ((0,p'),\bar{q}',v') \end{cases} \tag{5.1}$$

$$(M,\nu) \xrightarrow{d} (M',\nu') \quad ssi \quad \begin{cases} ((0,p),\bar{q},v) \xrightarrow{d} ((0,p'),\bar{q}',v') \text{ et} \\ (M',\nu') \approx ((0,p'),\bar{q}',v') \end{cases} \tag{5.2}$$

$\square$

Preuve. La preuve de ce théorème est en annexe A.1. $\square$

[2] (s,p) est l'état de SU où p est le vecteur donnant le nombre de jetons par place ; $\overline{q}$ donne l'état discret de $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ (donc $\bar{q}_i$ est l'état de $\mathcal{A}_i$) et v les valeurs des horloges $x_i, i \in [1..n]$ de chaque $\mathcal{A}_i$.

5.2.3 Conséquences des résultats précédents

La méthode précédemment décrite s'étend très simplement aux contraintes strictes.

Les variables utilisées dans les automates codent le marquage du réseau. Etant donné qu'une variable bornée peut être encodée par un automate, un automate temporisé avec variables bornées peut être encodé par un automate temporisé classique. Nous obtenons donc le théorème suivant :

Théorème 2 *Pour tout TPN $\mathcal{N} \in$ B-$\mathcal{TPN}(\leq,\geq)$, il existe un TA $\mathcal{A} \in \mathcal{TA}(\leq,\geq)$ tel que $\mathcal{N} \approx_{\mathcal{W}} \mathcal{A}$, et donc B-$\mathcal{TPN}(\leq,\geq) \precapprox_{\mathcal{W}} \mathcal{TA}(\leq,\geq)$. De plus B-$\mathcal{TPN} \precapprox_{\mathcal{W}} \mathcal{TA}$.*

Corollaire 1 *B-$\mathcal{TPN} \leq_{\mathcal{L}} \mathcal{TA}$ et B-$\mathcal{TPN}(\leq,\geq) \leq_{\mathcal{L}} \mathcal{TA}(\leq,\geq)$*

Corollaire 2 *Le problème du vide (langage vide d'un TPN) est décidable pour B-$\mathcal{TPN}$.*

Corollaire 3 *TCTL est décidable pour B-$\mathcal{TPN}$.*

Concernant la décidabilité de TCTL, nous en donnerons une preuve directe dans le chapitre suivant.

5.3 Relation d'ordre strict en terme de bisimulation

Nous prouvons maintenant que les $\mathcal{TPN}$ bornés sont strictement moins expressifs que les TA.

Considérons d'abord ce premier lemme sur les TPN.

Lemme 1 (Attendre ne désensibilise pas de transition) *Soit (M,ν) un marquage d'un TPN. Si $(M,\nu) \xrightarrow{t_1 t_2 \cdots t_k}$ avec $t_1 t_2 \cdots t_k$ une séquence de tir instantanée et $(M,\nu) \xrightarrow{d} (M_d,\nu_d)$ pour $d \geq 0$, alors $(M_d,\nu_d) \xrightarrow{t_1 t_2 \cdots t_k}$.*

Théorème 3 *Il n'existe pas de TPN temporellement bisimilaire (faiblement) à $\mathcal{A}_0 \in \mathcal{TA}$ de la figure (figure 5.3).*

Preuve. La preuve s'appuie sur le lemme précédent.

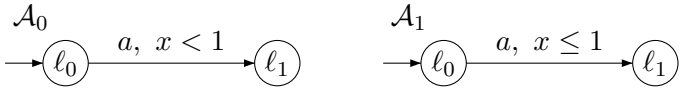


FIG. 5.3 – Automates temporisés $\mathcal{A}_0$ et $\mathcal{A}_1$

Supposons qu'il existe un TPN $\mathcal{N}$ temporellement bisimilaire (faiblement) à $\mathcal{A}_0$ et soit $\approx$ une relation de bisimulation temporelle faible entre $S_{\mathcal{N}}$ et $S_{\mathcal{A}_0}$. Soient $(M_0,\overline{0})$ l'état initial de $S_{\mathcal{N}}$ et $(\ell_0, v(x)=0)$ l'état initial de $S_{\mathcal{A}_0}$. Il existe une exécution (run) de durée 1 dans $S_{\mathcal{A}_0}$ conduisant à la configuration $(\ell_0,1)$ et donc il existe une exécution[3] : $(M_0,\overline{0}) \xrightarrow{\varepsilon^{i_0} d_1 \varepsilon^{i_1} d_2 \varepsilon^{i_2} \cdots d_n \varepsilon^{i_n}} (M_1,\nu_1)$ dans $S_{\mathcal{N}}$, avec $i_k \geq 1$ pour $1 \leq k \leq n-1$, $i_0 \geq 0$, $i_n \geq 0$ et $\sum_{1\leq k\leq n} d_k = 1$. Nous pouvons aussi supposer $d_k > 0$ pour tout k, et $i_n = 0$ car la configuration atteinte après

[3]la notation ε^k signifie classiquement ε à la puissance k.

d_n est aussi bisimilaire à $(\ell_0, 1)$. Nous avons $(M_0, \overline{0}) \xrightarrow{\varepsilon^{i_0} d_1 \varepsilon^{i_1} d_2 \varepsilon^{i_2} \cdots d_{n-1} \varepsilon^{i_{n-1}}} (M', \nu')$, tel que (M', ν') est bisimilaire à une configuration (ℓ_0, d') avec $d' = 1 - d_n < 1$. Cela implique que $(M', \nu') \xrightarrow{\varepsilon^* a}$. Or $(M', \nu') \xrightarrow{d_n} (M_1, \nu_1)$, donc, d'après le lemme 1, nous avons $(M_1, \nu_1) \xrightarrow{\varepsilon^* a}$ ce qui contredit $(M_1, \nu_1) \approx (\ell_0, 1)$ à partir duquel a ne peut pas être tirée. □
Ce résultat est également vrai pour des contraintes larges :

Théorème 4 *Il n'existe pas de TPN temporellement bisimilaire (faiblement) à $\mathcal{A}_1 \in \mathcal{TA}(\leq, \geq)$ de la figure (figure 5.3).*

Preuve. Supposons qu'il existe un TPN $\mathcal{N}$ temporellement bisimilaire (faiblement) à $\mathcal{A}_1$. Puisque $(\ell_0, 0) \xrightarrow{1} (\ell_0, 1)$, nous avons $(M_0, \overline{0}) \xrightarrow{1}_{\varepsilon} (M_1, \nu_1)$, tel que $(\ell_0, 1)$ et (M_1, ν_1) sont temporellement bisimilaire (faiblement). Etant donné que a peut être tiré à partir de $(\ell_0, 1)$, alors $(M_1, \nu_1) \xrightarrow{\varepsilon^* a}$ et $\xrightarrow{\varepsilon^* a}$ peut être tirée à partir de toutes les configurations (M_1', ν_1') accessibles à partir de (M_1, ν_1) en un temps nul (par des transitions ε). Il existe nécessairement une de ces configurations (M', ν') à partir de laquelle l'écoulement d'un temps $d > 0$ est possible conduisant à (M'', ν''). D'après le lemme 1, nous avons alors $(M'', \nu'') \xrightarrow{\varepsilon^* a}$. Or (M'', ν'') est faiblement temporellement bisimilaire à la configuration $(\ell_0, 1 + d)$ qui interdit le tir de a ; d'où la contradiction. □

A partir des théorèmes 2, 3 et 4, nous obtenons :

Corollaire 4 *La classe des TPN bornés est d'expressivité strictement inférieure à celle de la classe des TA : B-$\mathcal{TPN}$ $\precnsim_{\mathcal{W}}$ $\mathcal{TA}$. De plus, il en est de même pour ces classes réduites aux intervalles fermés : B-$\mathcal{TPN}(\leq, \geq)$ $\precnsim_{\mathcal{W}}$ $\mathcal{TA}(\leq, \geq)$.*

Etant donné ce résultat négatif, nous comparons maintenant l'expressivité des TPN et des TA relativement aux langages temporisés qu'ils acceptent.

5.4 Equivalence en termes d'acceptation de langage temporisé

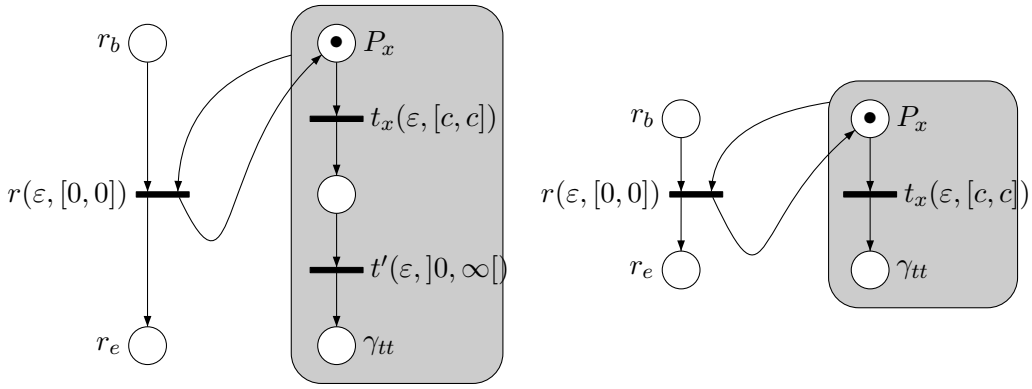
Dans cette section, nous prouvons que les TA et les TPN sont aussi expressifs du point de vue de l'acceptation de langage temporisé et nous donnons une traduction syntaxique des TA vers les TPN.

5.4.1 Traduction des TA vers les TPN préservant le langage

Soit $\mathcal{A} = (L, l_0, X, \Sigma_\varepsilon, E, Inv)$ un TA. Etant donné que nous ne nous intéressons qu'au langage accepté par $\mathcal{A}$, nous supposons que la fonction d'invariance Inv est uniformément vraie. Soit $\mathcal{C}_x$ l'ensemble des contraintes atomiques sur l'horloge x utilisées dans $\mathcal{A}$. Le réseau de Petri temporel résultant de notre traduction est construit à partir de blocs élémentaires modélisant la valeur de vérité des contraintes de $\mathcal{C}_x$. Dans ce réseau de Petri, les transitions sont notées $t(\ell, I)$ où t est le nom de la transition, $\ell \in, \Sigma_\varepsilon$ et I est l'intervalle de tir.

Encodage des contraintes atomiques. Soit $\varphi \in \mathcal{C}_x$ une contrainte atomique sur x. A partir de φ, nous définissons le bloc élémentaire $\mathcal{N}_\varphi$, décrit par les motifs des figures 5.4 ((a) et (b)) et 5.5.

Pour ne pas surcharger les figures, nous adoptons la sémantique suivante : les boîtes grises sont vues comme des macro-places ; un arc sortant du bord d'une boîte grise signifie qu'il y a



(a) Motif $\mathcal{N}_{x>c}$ (b) Motif $\mathcal{N}_{x\geq c}$ (avec $c > 0$)

FIG. 5.4 – Motifs pour $\mathcal{N}_{x>c}$ et $\mathcal{N}_{x\geq c}$

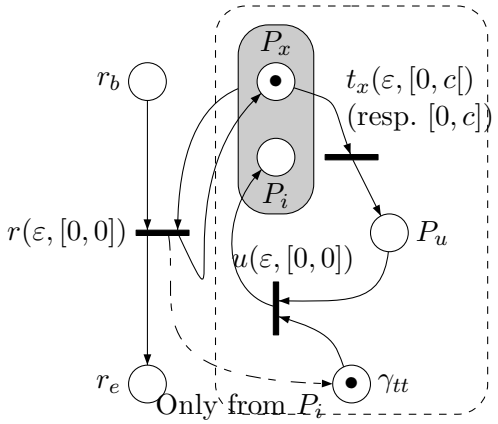


FIG. 5.5 – Motif $\mathcal{N}_{x<c}$ (resp. $\mathcal{N}_{x\leq c}$)

autant de copies de la transition (destination de l'arc) qu'il y a de places dans la boîte grise. Par exemple, le TPN de la figure 5.4.(b) aura deux copies de la transition r : une avec P_x et r_b comme places d'entrée et une avec r_b and γ_{tt} comme places d'entrée. Les deux copies de r auront toutes les deux r_e et P_x comme places destination. Notons le cas particulier du motif de la figure 5.5 pour lequel l'arc vers γ_{tt} n'est présent que pour la copie de r issue de la place P_i . En outre, nous supposons que l'automate $\mathcal{A}$ n'a pas de contrainte $x \geq 0$ (qui sont toujours vraies et peuvent donc être retirées), c'est pourquoi le motif de la figure 5.4.(b) est restreint à $c > 0$.

Chacun de ces motifs est constitué d'une partie « contrainte » (en gris pour la figure 5.4 et dans la zone pointillée pour la figure 5.5) qui modélise la valeur de vérité de la contrainte, et d'une autre partie « reset » utilisée pour la mise à jour du motif lorsque l'horloge x est remise à zéro.

La partie « contrainte » comprend une place γ_{tt} dont le marquage peut être interprété par : quand un jeton est disponible dans cette place, la contrainte atomique φ correspondante est vraie.

Lorsqu'une horloge x est remise à zéro, les parties grises des blocs doivent retourner à leur marquage *initial* : un jeton dans P_x pour la figure 5.4 et un jeton dans P_x et dans γ_{tt} pour

la figure 5.5. La stratégie pour cela est de mettre un jeton dans la place r_b (r_b signifie « reset begin »). Le temps ne peut alors pas s'écouler car il y a nécessairement un jeton dans toutes les places amonts d'une des copies de r qui est par conséquent sensibilisée et devra être tirée en temps nul.

Mise à zéro des horloges. Afin de remettre à zéro les blocs modélisant les contraintes sur une horloge x, nous les chaînons dans un ordre arbitraire : la place r_e du i^{eme} bloc est reliée à la place r_b du $i+1^{eme}$ bloc, *via* une transition ε de 0 unité de temps. Cela est illustré figure 5.6 pour les horloges x_1 and x_n. Considérons $R \subseteq X$, un ensemble non vide d'horloges. Soit $D(R)$, l'ensemble de contraintes atomiques sur les horloges de R. Nous notons $D(R) = \{\varphi_1^{x_1}, \varphi_2^{x_1}, \cdots, \varphi_{q_1}^{x_1}, \cdots, \varphi_{q_n}^{x_n}\}$ tel que $\varphi_i^{x_j}$ est la i^{eme} contrainte de l'horloge x_j. Pour mettre à jour tous les blocs de $D(R)$, nous connectons les chaînes de remise à zéro comme indiqué sur la figure 5.6. Le bloc dans la boîte pointillée est noté $\mathcal{N}_{Reset(R)}$: $r_b(R)$ est sa première place et $r_e(R)$ sa dernière. Pour mettre à jour la valeur de vérité des blocs de $D(R)$, il suffit de mettre un jeton dans $r_b(R)$. En un temps nul, le jeton arrivera à $r_e(R)$ en effectuant la mise à zéro de tous les blocs de $D(R)$.

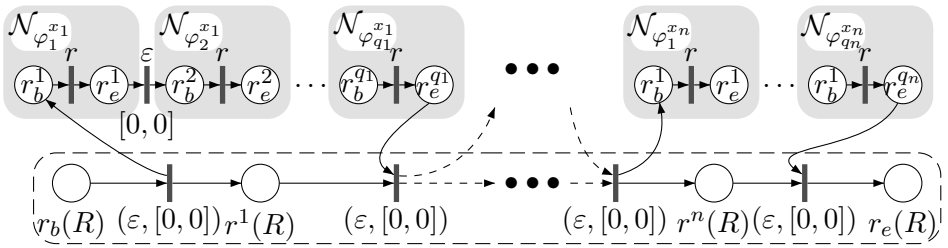


Fig. 5.6 – Motif $\mathcal{N}_{Reset(R)}$ pour la mise à zéro des motifs des blocs des horloges x_i, $1 \leq i \leq n$

La construction complète. D'abord nous créons une place P_ℓ pour chaque localité $\ell \in L$. Ensuite, nous construisons les blocs $\mathcal{N}_\varphi$ pour toutes les contraintes atomiques φ qui apparaissent dans $\mathcal{A}$. Enfin, pour chaque $R \subseteq X$ tel qu'il existe un arc $e = (\ell, \gamma, a, R, \ell') \in E$, nous construisons le bloc $\mathcal{N}_{Reset(R)}$.

Pour tout arc $(\ell, \gamma, a, R, \ell') \in E$ avec $\gamma = \wedge_{i=1,n}\varphi_i$ and $n \geq 0$ nous procédons comme suit :
1. création d'une transition $f(a, [0, \infty[)$,
2. si $n \geq 1$, création d'une transition $r(\varepsilon, [0, 0])$,
3. connexions de $f(a, [0, \infty[)$ et $r(\varepsilon, [0, 0])$ aux places γ_{tt} des blocs $\mathcal{N}_{\varphi_i}$, aux places P_ℓ et P'_ℓ ainsi qu'aux places $r_b(R)$ et $r_e(R)$ du bloc $\mathcal{N}_{Reset(R)}$ comme indiqué sur la figure 5.7. Dans le cas où $\gamma = \texttt{true}$ ($n = 0$), la place P_ℓ est la seule place d'entrée de $f(a, [0, \infty[)$. Dans le cas où $R = \emptyset$, il n'y a pas de transition $r(\varepsilon, [0, 0])$ et la place de sortie de $f(a, [0, \infty[)$ est directement $P_{\ell'}$.

Pour terminer la construction, il suffit de mettre un jeton dans la place P_{ℓ_0} si ℓ_0 est la localité initiale de l'automate, et de mettre chaque bloc $\mathcal{N}_\varphi$ dans son marquage initial.

Nous notons $\Delta(\mathcal{A})$ le TPN obtenu. Remarquons que par construction :
1) $\Delta(\mathcal{A})$ est sauf (1-borné),
2) dans tout marquage accessible M de $\Delta(\mathcal{A})$, nous avons $\sum_{\ell \in L} M(P_\ell) \leq 1$.

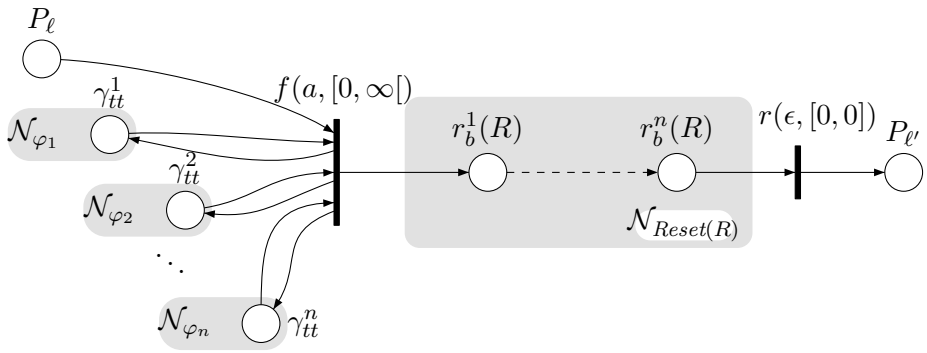


FIG. 5.7 – Motif $\mathcal{N}_e$ d'un arc $e = (\ell, \gamma, a, R, \ell')$

Un bloc associé à une contrainte atomique a la taille fixe du motif correspondant. Un bloc associé à la remise à zéro d'une horloge a une taille linéaire en fonction du nombre de contraintes atomiques sur cette horloge. Un bloc associé à un arc a une taille linéaire en fonction de la taille de la description de l'arc. Par conséquent, la taille de $\Delta(\mathcal{A})$ est linéaire en fonction de la taille de $\mathcal{A}$ ce qui améliore la complexité quadratique de la traduction (restreinte) de [HKSLT02].

Enfin, pour prouver que $\mathcal{L}(\Delta(\mathcal{A})) = \mathcal{L}(\mathcal{A})$, nous construisons deux relations de simulation $\sqsubseteq_1$ et $\sqsubseteq_2$ tel que $\Delta(\mathcal{A}) \sqsubseteq_1 \mathcal{A}$ et $\mathcal{A} \sqsubseteq_2 \Delta(\mathcal{A})$.

5.4.2 $\Delta(\mathcal{A})$ et $\mathcal{A}$ acceptent le même langage temporisé

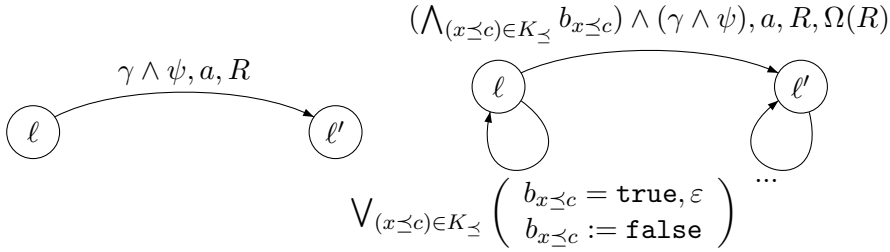
Nous allons maintenant prouver le théorème suivant :

Théorème 5 *Soient $\mathcal{A}$ un TA et $\Delta(\mathcal{A})$ sa traduction en TPN telle que définie précédemment, nous avons $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\Delta(\mathcal{A}))$.*

Preuve. La preuve complète est en annexe A.2. Nous en donnons ici les éléments principaux.

Nous prouvons d'abord que $\Delta(\mathcal{A})$ simule (faiblement) $\mathcal{A}$ ce qui implique $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\Delta(\mathcal{A}))$. Ensuite nous montrons que nous pouvons définir un TA $\mathcal{A}'$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ et $\mathcal{A}'$ simule (faiblement) $\Delta(\mathcal{A})$ ce qui implique $\mathcal{L}(\Delta(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. Il est suffisant de prouver cela pour le cas où $\mathcal{A}$ n'a pas de transition ε. En effet, dans le cas contraire, nous pouvons renommer les transitions ε en une nouvelle lettre $\mu \notin \Sigma_\varepsilon$ et obtenir ainsi un automate $\mathcal{A}_\mu$ sans transition ε. Nous appliquons alors notre traduction à $\mathcal{A}_\mu$ et obtenons un TPN dans lequel nous remplaçons les étiquettes μ par ε.

Rappelons que $\mathcal{A} = (L, \ell_0, X, \Sigma_\varepsilon, E, Inv)$ et $\Delta(\mathcal{A}) = (P, T, \Sigma_\varepsilon, {}^\bullet(.), (.)^\bullet, M_0, \Lambda, I, F_\Delta, R_\Delta)$ et notons $X = \{x_1, \cdots, x_k\}$, $P = \{p_1, \cdots, p_m\}$ et $T = \{t_1, \cdots, t_n\}$. L'ensemble des contraintes atomiques de $\mathcal{A}$ est $\mathcal{C}_\mathcal{A}$. La place γ_{tt} d'un bloc $\mathcal{N}_{x \bowtie c}$ (pour $x \bowtie c$: une contrainte atomique de $\mathcal{A}$) est notée $\gamma_{tt}^{x \bowtie c}$.



(a) Arc $(\ell, \gamma \wedge \psi, a, R, \ell')$ dans $\mathcal{A}$ (b) Arc étendu dans $\mathcal{A}'$.

Fig. 5.8 – De $\mathcal{A}$ vers $\mathcal{A}'$.

Preuve de $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\Delta(\mathcal{A}))$. Nous prouvons pour cela que $\Delta(\mathcal{A})$ simule $\mathcal{A}$. Nous définissons la relation $\sqsubseteq \subseteq (L \times \mathbb{R}^n_{\geq 0}) \times (\mathbb{N}^p \times \mathbb{R}^m_{\geq 0})$ par :

$$(\ell, v) \sqsubseteq (M, \nu) \iff \begin{cases} (1)\, M(P_\ell) = 1 \\ (2)\, \textit{pour tout } \varphi = x \bowtie c, \bowtie \in \{<, \leq\},\ M(P_u) = 0 \\ (3)\, \textit{pour tout} \varphi \in \mathcal{C}_\mathcal{A},\ v \in [\![\varphi]\!] \iff M(\gamma^\varphi_{tt}) = 1 \end{cases} \tag{5.3}$$

La preuve que $\sqsubseteq$ est une relation de simulation faible de $\mathcal{A}$ par $\Delta(\mathcal{A})$ figure dans l'annexe A.2.

Preuve de $\mathcal{L}(\Delta(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A})$. Pour cela, nous ne pouvons pas montrer une simulation évidente de $\Delta(\mathcal{A})$ par $\mathcal{A}$. En effet, dans $\Delta(\mathcal{A})$, les blocs $\mathcal{N}_{x \bowtie c}$ avec $\bowtie \in \{<, \leq\}$, imposent une décision sur le tir de la transition t_x (et u immédiatement après). Cela revient à décider qu'à partir d'un certain point $x \bowtie c$ devient faux et le restera jusqu'à la prochaine mise à zéro de l'horloge x. Pour établir une relation de simulation, nous allons construire un TA $\mathcal{A}'$ qui accepte le même langage que $\mathcal{A}$ mais ayant la possibilité de décider parfois (de manière non-deterministe) de ne pas utiliser une transition avec une garde $x \bowtie c$ (jusqu'au prochain reset de x). Il est alors possible de construire une relation de simulation de $\Delta(\mathcal{A})$ par $\mathcal{A}'$.

Nous notons $\preceq$ pour $\{<, \leq\}$ et $\succeq$ pour $\{>, \geq\}$. Soit $K_\preceq$ l'ensemble des contraintes $x \preceq c$ in $\mathcal{A}$. Pour chaque $x \preceq c \in K_\preceq$, nous introduisons une variable booléenne $b_{x \preceq c}$ initialement à vrai.

La construction de $\mathcal{A}'$ commence par $\mathcal{A}' = \mathcal{A}$. Nous enrichissons ensuite $\mathcal{A}'$ comme indiqué sur la figure 5.8. Soit $(\ell, \gamma \wedge \psi, a, R, \ell')$ un arc de $\mathcal{A}'$ avec $\gamma = \wedge_{x \preceq c \in K_\preceq} x \preceq c$ et $\psi = \wedge_{x \succeq c \in K_\succeq} x \succeq c$. Pour ces arcs, nous renforçons[4] les gardes $\gamma \wedge \psi$ pour obtenir : $\gamma' = \gamma \wedge \psi \wedge \bigwedge_{x \preceq c \in K_\preceq} b_{x \preceq c}$. Ainsi, la transition $(\ell, \gamma \wedge \psi, a, R, \ell')$ peut être tirée dans $\mathcal{A}'$ seulement si la garde correspondante dans $\mathcal{A}$ et la conjonction des $b_{x \preceq c}$ est vraie. Nous remettons également à vrai les variables $b_{x \preceq c}$ dont $x \in R$ dans la transition $(\ell, \gamma \wedge \psi, a, R, \ell')$ et nous notons $\Omega(R) = \wedge_{x \in R} b_{x \preceq c} := \texttt{true}$.

Soit ℓ une localité de $\mathcal{A}'$. Pour chaque variable $b_{x \preceq c}$, nous ajoutons une boucle $(\ell, b_{x \preceq c} = \texttt{true}, \varepsilon, b_{x \preceq c} := \texttt{false}, \ell)$ dans $\mathcal{A}'$, ce qui signifie que l'automate $\mathcal{A}'$ peut décider de manière non-deterministe[5] de mettre $b_{x \preceq c}$ à faux (voir figure 5.8). Il y a autant de boucles que de variables $b_{x \preceq c}$ ce qui est représenté par un $\bigvee$ dans la figure 5.8. Cet automate non-deterministe $\mathcal{A}'$ accepte le même langage temporisé que $\mathcal{A}$: $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

[4] Nous avons besoin de TA étendus avec des variables booléennes ; cela n'ajoute rien au pouvoir d'expression des TA.

[5] Cela ajoute des transitions ε à $\mathcal{A}'$ mais la restriction que nous avons faite au début - $\mathcal{A}$ ne possède pas de transition ε - est inutile pour montrer que $\mathcal{A}'$ simule faiblement $\Delta(\mathcal{A})$.

Nous pouvons maintenant construire la relation de simulation de $\Delta(\mathcal{A})$ par $\mathcal{A}'$. Notons (ℓ, v, b) une configuration de $\mathcal{A}'$ avec b le vecteur des variables b_φ. Nous définissons la relation $\sqsubseteq \subseteq (\mathbb{N}^p \times \mathbb{R}^m_{\geq 0}) \times (L \times \mathbb{R}^n_{\geq 0} \times \mathbb{B}^k)$ par :

$$(M,\nu) \sqsubseteq (\ell, v, b) \iff \begin{cases} (1)\, M(P_\ell) = 1 \\ (2)\, \forall \varphi = x > c \in K_>,\ v \in [\![\varphi]\!] \iff M(\gamma^\varphi_{tt}) = 1 \\ (3)\, \forall \varphi = x \geq c \in K_\geq,\ v \in [\![\varphi]\!] \iff M(\gamma^\varphi_{tt}) = 1 \vee \\ \qquad\qquad (M(P^\varphi_x) = 1 \wedge \nu(t^\varphi_x) = c) \\ (4) \forall \varphi \in K_\preceq, M(P^\varphi_i) = 1 \iff (b_\varphi = \texttt{false} \vee v \notin [\![\varphi]\!]) \end{cases} \tag{5.4}$$

La preuve que $\sqsubseteq$ est une relation de simulation faible de $\Delta(\mathcal{A})$ par $\mathcal{A}'$ est dans l'annexe A.2.

Nous avons donc $\mathcal{A}'$ simule $\Delta(\mathcal{A})$ ainsi $\mathcal{L}(\Delta(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}')$ et donc $\mathcal{L}(\Delta(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A})$.

Nous pouvons donc conclure que $\mathcal{L}(\Delta(\mathcal{A})) = \mathcal{L}(\mathcal{A})$, ce qui termine la preuve du théorème 5. □

5.4.3 Conséquences des résultats précédents

Notons que le théorème 5 ainsi que les résultats qui suivent s'étendent facilement aux langages temporisés généraux (langage temporisé de Büchi) comme nous l'avons fait dans [BCH+05b].

D'après les résultats précédents, nous obtenons les corollaires suivants :

Corollaire 5 *Les classes B-$\mathcal{TPN}$ et $\mathcal{TA}$ sont d'expressivité égale en terme d'acceptation de langage temporisé : B-$\mathcal{TPN}$ $=_\mathcal{L}$ $\mathcal{TA}$ et B-$\mathcal{TPN}(\leq,\geq)$ $=_\mathcal{L}$ $\mathcal{TA}(\leq,\geq)$.*

Preuve. D'après le corollaire 1, nous savons que B-$\mathcal{TPN}$ $\leq_\mathcal{L}$ $\mathcal{TA}$. Le théorème 5 prouve que $\mathcal{TA}$ $\leq_\mathcal{L}$ $\mathcal{TPN}$ et par conséquent B-$\mathcal{TPN}$ $=_\mathcal{L}$ $\mathcal{TA}$. La traduction précédente peut être restreinte aux contraintes larges et par le même raisonnement on obtient B-$\mathcal{TPN}(\leq,\geq)$ $=_\mathcal{L}$ $\mathcal{TA}(\leq,\geq)$. □

Corollaire 6 *Les classes 1-B-$\mathcal{TPN}$ et B-$\mathcal{TPN}$ sont d'expressivité égale en terme d'acceptation de langage temporisé :* 1-B-$\mathcal{TPN}$ $=_\mathcal{L}$ B-$\mathcal{TPN}$ $=_\mathcal{L}$ $\mathcal{TA}$ *et* 1-B-$\mathcal{TPN}(\leq,\geq)$ $=_\mathcal{L}$ B-$\mathcal{TPN}(\leq,\geq)$ $=_\mathcal{L}$ $\mathcal{TA}(\leq,\geq)$.

Preuve. Soit $\mathcal{N} \in$ B-$\mathcal{TPN}$. D'après le théorème 2, il existe un TA $A_\mathcal{N}$ tel que $\mathcal{L}(\mathcal{N}) = \mathcal{L}(A_\mathcal{N})$. D'après le théorème 5 et en utilisant la traduction correspondante nous obtenons $\Delta(A_\mathcal{N}) \in$ 1-B-$\mathcal{TPN}$ tel que $\mathcal{L}(A_\mathcal{N}) = \mathcal{L}(\Delta(A_\mathcal{N}))$.

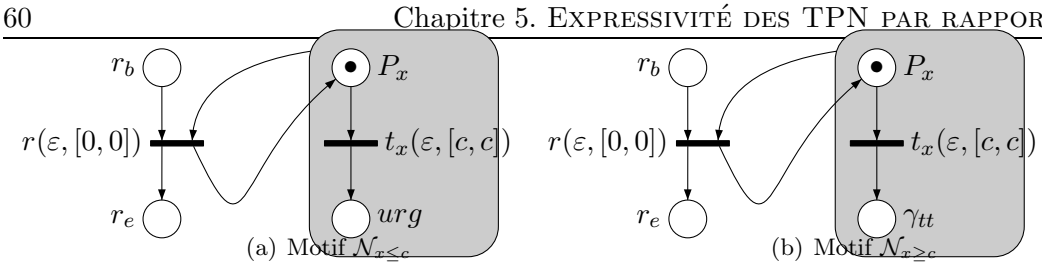
□

Corollaire 7 *Le problème de l'universalité (langage universel d'un TPN) est indécidable pour B-$\mathcal{TPN}$ (et aussi pour 1-B-$\mathcal{TPN}$).*

5.5 Sous-classe des TA équivalente par bisimulation aux TPN « à la Merlin »

5.5.1 Définition de la sous-classe $\mathcal{TA}_{tpn}(\leq,\geq)$

Dans cette section, nous nous intéressons à la classe des TA équivalente (en terme de bisimulation) aux TPN « à la Merlin » bornés c'est-à-dire B-$\mathcal{TPN}(\leq,\geq)$ (avec des transitions ε).



(a) Motif $\mathcal{N}_{x\leq c}$ (b) Motif $\mathcal{N}_{x\geq c}$

Fig. 5.9 – Motifs $\mathcal{N}_{x\leq c}$ et $\mathcal{N}_{x\geq c}$

D'après les théorèmes 3 et 4, nous identifions que les TPN ne peuvent pas modéliser le fait qu'attendre peut rendre impossible le tir d'une transition. Ce résultat est confirmé en regardant le résultat de la traduction des TPN vers TA de la section 5.2. Nous pouvons en effet constater que le TA obtenu a la particularité suivante : les gardes sont seulement de la forme $x \geq c$ et les invariants $x \leq c$. Un autre élément intéressant de cette traduction est que, dans les TA obtenus, entre 2 mises à zéro d'une horloge x, les contraintes d'invariant sur cette horloge vont en s'élargissant.

Nous notons $\varphi \in Inv(\ell)$ pour les contraintes atomiques de l'invariant d'une localité ℓ c'est-à-dire lorsque $\varphi = (x \leq c) \in \{\varphi_1, \varphi_2 \cdots \varphi_n\}$ tel que $Inv(\ell) = \wedge_{k=1,n}\varphi_k$.

Nous allons donc considérer la sous-classe syntaxique des automates temporisés $\mathcal{TA}_{tpn}(\leq, \geq)$ définie par :

Définition 22 (Sous classe $\mathcal{TA}_{tpn}(\leq, \geq)$) *La sous-classe $\mathcal{TA}_{tpn}(\leq, \geq)$ est définie par l'ensemble des TA $(X, \Sigma, E, Inv, F, R)$ avec :*
1) les contraintes sont des contraintes larges : les gardes sont des conjonctions de contraintes atomiques de la forme $x \geq c$ et les invariants sont des conjonctions de contraintes atomiques de la forme $x \leq c$,
2) les invariants sont ordonnés *: c'est-à-dire $\forall e = (\ell, \gamma, a, R, \ell') \in E$, $\forall x \notin R$ tel que $x \leq c$ est une contrainte atomique de $Inv(\ell)$, si $(x \leq c') \in Inv(\ell')$ alors $c' \geq c$*

Nous allons maintenant proposer une traduction des automates de la sous classe $\mathcal{TA}_{tpn}(\leq, \geq)$ vers les TPN « à la Merlin » préservant la bisimulation.

5.5.2 La traduction

Pour un automate $\mathcal{A} \in \mathcal{TA}_{tpn}(\leq, \geq)$, nous reprenons la construction de la section 5.4 (pour le langage) en changeant le motif de $\mathcal{N}_{x\leq c}$ qui devient le motif de l'invariant comme indiqué sur la figure 5.9.a. Le motif $\mathcal{N}_{x\geq c}$ pour les gardes est rappelé figure 5.9.b. Les blocs pour les reset sont ceux de la section 5.4.

La construction complète. De la même manière que dans la section 5.4, nous créons d'abord une place P_ℓ pour chaque localité $\ell \in L$. Ensuite, nous construisons les blocs $\mathcal{N}_\varphi$ pour toutes les contraintes atomiques $\varphi = (x \geq c)$ (figure 5.9.b) qui apparaissent dans les gardes de $\mathcal{A}$ et pour toutes les contraintes atomiques $\varphi = (x \leq c)$ (figure 5.9.a) qui apparaissent dans les invariants de $\mathcal{A}$. Enfin, pour chaque $R \subseteq X$ tel qu'il existe un arc $e = (\ell, \gamma, a, R, \ell') \in E$, nous construisons le bloc $\mathcal{N}_{Reset(R)}$ (figure 5.6). Pour tout arc $(\ell, \gamma, a, R, \ell') \in E$, nous procédons de la même manière que dans la section langage comme indiqué sur la figure 5.10.a.

Pour toute localité $\ell \in L$ avec $Inv(\ell) = \wedge_{k=1,n}\mathcal{I}_k$ nous procédons ainsi :
1. si $n \geq 1$, création $\forall k \leq n$ d'une transition $I_k(\varepsilon, [0, 0])$,

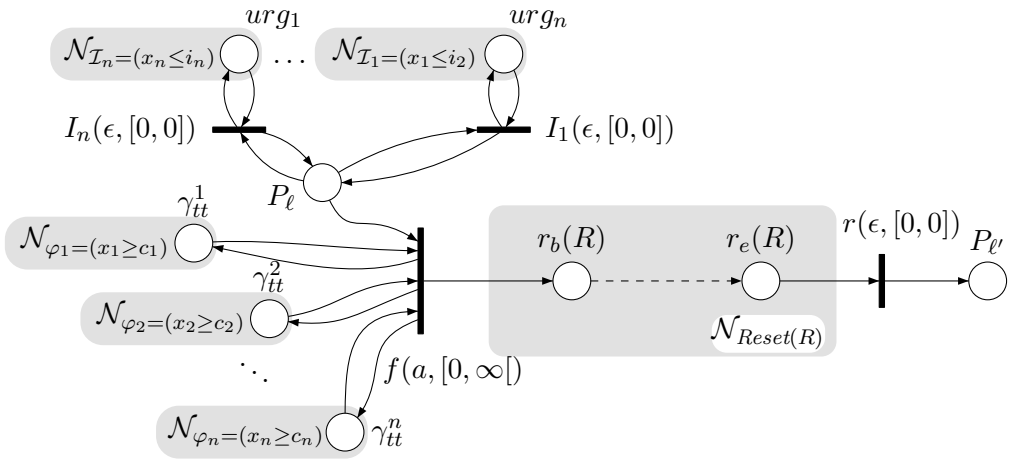


FIG. 5.10 – Motif $\mathcal{N}_e$ d'un arc $e=(\ell,\gamma,a,R,\ell')$

2. connexions de $f(a,[0,\infty[)$ et $I_i(\varepsilon,[0,0])$ à P_ℓ et aux places urg des blocs $\mathcal{N}_{\mathcal{I}_k}$, comme indiqué sur la figure 5.10.

Rappelons que $\mathcal{A}=(L,\ell_0,X,\Sigma_\varepsilon,E,Inv)$ et $\Delta(\mathcal{A})=(P,T,\Sigma_\varepsilon,{}^\bullet(.),(.)^\bullet,M_0,\Lambda,I)$ et notons $X=\{x_1,\cdots,x_k\}$, $P=\{p_1,\cdots,p_m\}$ et $T=\{t_1,\cdots,t_n\}$. L'ensemble des contraintes atomiques de $\mathcal{A}$ est $\mathcal{C}_\mathcal{A}$ et l'ensemble des contraintes $x\bowtie c$ dans $\mathcal{A}$ est $K_{\bowtie}$ (ainsi $\mathcal{C}_\mathcal{A}=K_{\geq}\cup K_{\leq}$). La place P_x et la transition t_x d'un bloc $\mathcal{N}_\varphi$ (pour φ : une contrainte atomique (garde ou invariant) de $\mathcal{A}$) sont notées respectivement P^φ_x et t^φ_x. La place γ_{tt} d'un bloc $\mathcal{N}_{x\geq c}$ (garde) est notée $\gamma^{x\geq c}_{tt}$. La place urg d'un bloc $\mathcal{N}_{x\leq c}$ (invariant) est notée $urg^{x\leq c}$.

5.5.3 Equivalence en terme de bisimulation.

Nous notons $\Delta(\mathcal{A})$ le TPN obtenu par la traduction précédente. Remarquons que, par construction, le TPN obtenu $\Delta(\mathcal{A})$ est sauf (1-borné).

Nous pouvons maintenant construire la relation de bisimulation entre $\Delta(\mathcal{A})$ et $\mathcal{A}$. Soit (ℓ,v) une configuration de $\mathcal{A}$ et (M,ν) une configuration de $\Delta(\mathcal{A})$. Nous définissons la relation $\approx\subseteq(\mathbb{N}^p\times\mathbb{R}^m_{\geq 0})\times(L\ \times\mathbb{R}^n_{\geq 0})$ par :

$$(M,\nu)\approx(\ell,v)\iff\begin{cases}(1)\,M(P_\ell)=1\\(2)\,\forall\varphi\in\mathcal{C}_\mathcal{A},\nu(t^\varphi_x)=v(x)\\(3)\,\forall\varphi=(x\geq c)\in K_{\geq},\,v\in[\![\varphi]\!]\iff M(\gamma^\varphi_{tt})=1\vee\\\qquad\qquad(M(P^\varphi_x)=1\wedge\nu(t^\varphi_x)=c)\\(4)\,\forall\varphi=(x\leq c)\in Inv(\ell),\,v\in[\![\varphi]\!]\iff\\\qquad\qquad M(P^\varphi_x)=1\vee\\\qquad\qquad(M(urg^\varphi)=1\wedge\nu(t^\varphi_x)=c)\end{cases}\tag{5.5}$$

Remarquons que le point 2 de cette équation est vrai même lorsque la transition t^φ_x n'est pas sensibilisée.

Théorème 6 *Soient $\mathcal{A}$ un TA appartenant à la classe $\mathcal{TA}_{tpn}(\leq,\geq)$ et $\Delta(\mathcal{A})$ sa traduction en TPN telle que définie précédemment, nous avons $\Delta(\mathcal{A})\approx\mathcal{A}$ où $\approx$ est une relation de bisimulation faible.*

Preuve. La preuve de ce théorème est dans l'annexe A.3 □

5.5.4 Conséquences des résultats précédents

D'après les résultats précédents, nous obtenons les corollaires suivants :

Corollaire 8 *Les classes B-$\mathcal{TPN}(\leq,\geq)$ et $\mathcal{TA}_{tpn}(\leq,\geq)$ sont d'expressivité égale en terme de bisimulation : B-$\mathcal{TPN}(\leq,\geq) \approx_{\mathcal{W}} \mathcal{TA}_{tpn}(\leq,\geq)$.*

Preuve. D'après le théorème 6, tout TA $\mathcal{A} \in \mathcal{TA}_{tpn}(\leq,\geq)$ peut être traduit en un TPN $\Delta_1(\mathcal{A}) \in$ 1-B-$\mathcal{TPN}(\leq,\geq)$ (et donc aussi dans B-$\mathcal{TPN}(\leq,\geq)$) qui lui est temporellement bisimilaire. D'après le théorème 2, tout TPN $\mathcal{N} \in$ B-$\mathcal{TPN}(\leq,\geq)$ peut être traduit en un TA $\Delta_2(\mathcal{N}) \in \mathcal{TA}_{tpn}(\leq,\geq)$ qui lui est temporellement bisimilaire. □

Corollaire 9 *Les classes 1-B-$\mathcal{TPN}(\leq,\geq)$ et B-$\mathcal{TPN}(\leq,\geq)$ sont d'expressivité égale en terme de bisimulation : 1-B-$\mathcal{TPN}(\leq,\geq) \approx_{\mathcal{W}}$ B-$\mathcal{TPN}(\leq,\geq) \approx_{\mathcal{W}} \mathcal{TA}_{tpn}(\leq,\geq)$.*

Preuve. Soit $\mathcal{N} \in$ B-$\mathcal{TPN}(\leq,\geq)$. D'après le théorème 2, il existe un TA $A_{\mathcal{N}}$ tel que $\mathcal{N} \approx_1 A_{\mathcal{N}}$. D'après le théorème 6 et en utilisant la traduction correspondante nous obtenons $\Delta(A_{\mathcal{N}}) \in$ 1-B-$\mathcal{TPN}$ tel que $A_{\mathcal{N}} \approx_2 (\Delta(A_{\mathcal{N}}))$.
□

5.6 Conclusion

Nous avons prouvé que tout TPN peut être traduit en un TA avec variables qui lui est temporellement bisimilaire et donc que tout TPN borné peut être traduit en un TA qui lui est temporellement bisimilaire. Nous avons prouvé que tout TA peut être traduit en un TPN acceptant le même langage temporisé mais qu'il existe un TA dont aucun TPN n'est temporellement bisimilaire. Les TA et les TPN bornés (ainsi que les TA avec variables non bornées et les TPN) ont donc la même expressivité en terme de langage temporisé (nous avons étendu ce résultat aux langages temporisés généraux (langages temporisés de Büchi) dans [BCH$^+$05b]). De plus, les TA sont strictement plus expressifs que les TPN bornés en terme de bisimulation. Par ailleurs, nous avons identifié la sous-classe des TA équivalente aux TPN en terme de bisimulation.

Ces résultats s'étendent très facilement aux extensions à chronomètres de ces deux modèles que nous étudierons dans le chapitre 7.

Les TPN généraux (non bornés) ont le pouvoir d'expression d'une machine de Turing ce qui n'est pas le cas des TA. Or les TPN bornés sont strictement moins expressifs que les TA en terme de bisimulation, par conséquent, les classes des TPN généraux et des TA sont incomparables.

Enfin nous avons déduit des traductions présentées dans ce chapitre un certain nombre de corollaires :

- les classes des TPN bornés et des TPN saufs ont la même expressivité en terme de langage temporisé,
- les classes des TPN bornés et des TPN saufs avec contraintes larges ont la même expressivité en terme de bisimulation,
- le problème du vide (langage vide) est décidable pour les TPN bornés,
- le problème de l'universalité (langage universel) est indécidable pour les TPN,

– TCTL est décidable pour les TPN bornés.

Concernant ce dernier point, la traduction des TPN vers les TA que nous avons proposée permet aussi, d'un point de vue pratique, la vérification des TPN en utilisant les outils sur les TA [CR04]. De plus cette première approche nous permet d'envisager d'adapter aux TPN les méthodes de model-checking mises au point sur les TA. Cela constitue l'objet du chapitre suivant.

Chapitre 6

Vérification de propriétés TCTL sur les réseaux de Petri temporels

But de l'étude. Étant donnée une question posée sur un système, un modèle peut être défini comme une abstraction du système telle que la réponse à la question est la même pour le système et son modèle. L'étape à laquelle nous nous intéressons ici est la conception d'algorithmes de vérification de propriétés sur un modèle particulier : les réseaux de Petri temporels. Dans [Lam77], Lamport classe les propriétés en deux catégories principales : les propriétés de *vivacité* qui assurent que sous certaines conditions, quelque chose finira par avoir lieu et les propriétés de sûreté qui expriment le fait que quelque chose de « mauvais » ne se produira jamais. Ces propriétés peuvent être exprimées dans différentes logiques. En particulier, nous nous intéressons aux logiques temporelles linéaire (LTL) et arborescente (CTL) et à l'extension temporisée de cette dernière : TCTL (*Timed Computation Tree Logic*) [ACD90].

Ces propriétés peuvent être vérifiées en explorant l'espace d'états du modèle, totalement ou en partie. Les horloges d'un TPN prenant leurs valeurs dans l'ensemble des réels, l'ensemble des états accessibles d'un TPN est en général infini, même lorsque le réseau est borné. Il est donc nécessaire de recourir à une méthode symbolique afin de calculer une abstraction finie de l'espace d'états.

Dans la littérature, les abstractions de l'espace d'états des réseaux de Petri temporels sont principalement basées sur la notion de classe d'états alors que celles des automates temporisés sont basées sur les régions et les zones. D'un point de vue chronologique, les classes d'états mises au point sur les TPN sont bien antérieures aux abstractions des automates temporisés. De même, les structures de données efficaces codant les contraintes linéaires entre les horloges - connues sous le nom de *difference bound matrices* (DBM) et proposées dans [BM83] pour les réseaux de Petri temporels - sont les mêmes que celles proposées à nouveau dans [Dil89] et utilisées par la suite pour les automates temporisés.

Malgré cela, le constat que l'on peut faire aujourd'hui est qu'aucune méthode directe[1] n'a été proposée pour la vérification de propriétés temporelles quantitatives sur les TPN. De plus, aucun outil de model-checking (c'est à dire permettant à partir du modèle d'un système S et d'une propriété φ, de décider $S \models \varphi$) n'est disponible alors que plusieurs méthodes et outils efficaces permettant la vérification de propriétés exprimées avec la logique temporelle TCTL existent sur les automates temporisés : Uppaal, Kronos.

[1] C'est-à-dire sans utilisation d'observateur ou sans traduction vers les automates temporisés.

Dans ce chapitre nous nous intéressons au model-checking de TCTL sur les TPN que nous qualifierons de natif c'est à dire sans passer par un TA intermédiaire[2]. Nous proposons un TCTL pour TPN (*TPN-TCTL*) pour lequel les opérateurs sont étendus avec un intervalle spécifiant une contrainte de temps sur une séquence de tirs. Nous prouvons la décidabilité de *TPN-TCTL* sur les TPN bornés et encadrons sa complexité. Nous proposons une méthode *en avant* et *à la volée* de vérification d'une sous classe de *TPN-TCTL* sur les TPN consistant à calculer et explorer une abstraction de l'espace d'états. Cette abstraction est basée sur la notion de zone et la méthode est inspirée de celle mise au point pour les automates temporisés et implémentée dans l'outil Uppaal. Comme pour les automates temporisés, cette abstraction peut nécessiter l'emploi d'un opérateur de surapproximation des zones (connue sous le nom d'opérateur de *normalisation* ou de *k-approximation*)pour assurer que son calcul finisse. Nous proposons une approximation plus fine que la *k-approximation* conduisant à une abstraction plus compacte et prouvons sa correction pour la classe de *TPN-TCTL* que nous considérons.

Contexte des travaux. Ces travaux ont été réalisés avec Guillaume Gardey et Hanifa Boucheneb pendant mon séjour à l'école Polytechnique de Montréal (de février à juillet 2005). Guillaume Gardey et moi-même avions été invité par Hanifa Boucheneb (une publication est en cours). Ces travaux correspondent à une partie de la thèse de Guillaume Gardey et ont été initiés par l'étude d'une abstraction de l'espace d'états d'un TPN basée sur les zones dans [GRR03, GRR06].

6.1 Introduction

Abstraction de l'espace d'états d'un TPN. Les horloges d'un TPN prenant leurs valeurs dans l'ensemble des réels, l'ensemble des états accessibles d'un TPN est en général infini, même lorsque le réseau est borné. Il est donc nécessaire de recourir à une méthode symbolique afin de calculer une abstraction fini de l'espace d'états.

On peut trouver plusieurs abstractions de l'espace d'états d'un TPN dans la littérature : le *graphe des classes d'états (State Class Graph : SCG)* [BD91], le *graphe des régions géométriques* (Geometric Region Graph : GRG) [YR98], le graphe des classes d'états linéaires fortes (*Strong State Class Graph* : SSCG) [BV03], le graphe des zones (*Zone Based Graph* : ZBG) que nous avons proposés dans [GRR03, GRR06] et le graphe des classes atomiques (*Atomic State Class Graphs* : ASCGs) [YR98, BV03, BH04].

Ces abstractions différent sur les classes de propriétés qu'elles préservent. Les graphes des classes fournissent une représentation finie de l'espace d'états d'un TPN borné préservant les propriétés LTL [BD91] ou CTL* [BV03, YR98, BH04]. Le graphe des régions géométriques préserve les propriétés CTL et le graphe des zones préserve l'accessibilité des marquages ou les propriétés LTL en fonction du critère de convergence utilisé.

A partir de ces abstractions, aucune méthode n'a été proposée[3] pour la vérification de propriétés temporelles quantitatives.

[2]En effet, la traduction structurelle proposée dans le chapitre précédent permet la vérification de propriétés exprimées avec la logique temporelle TCTL sur les TPN en les vérifiant sur le TA obtenu [CR04]. Nous avons également proposé une méthode permettant de traduire les TPN en TA avec une réduction du nombre d'horloges du TA obtenu afin de rendre la vérification plus efficace [LR03b].

[3]Exception faite de l'utilisation d'observateurs traduisant une propriété temporelle quantitative en un problème d'accessibilité de marquage.

$$
\begin{array}{lll}
(M,v) \models \mathit{GMEC} & \text{ssi} & M \models \mathit{GMEC} \\
(M,v) \not\models \mathbf{false} & & \\
(M,v) \models \neg\varphi & \text{ssi} & (M,v) \not\models \varphi \\
(M,v) \models \varphi \Rightarrow \psi & \text{ssi} & (M,v) \not\models \varphi \text{ or } (M,v) \models \psi \\
(M,v) \models \exists\varphi\, \mathcal{U}_I \psi & \text{ssi} & \exists\sigma \in [\![\mathcal{N}]\!] \ \textit{tel que} \ \begin{cases} (s_0,v_0) = (M,v) \\ \forall i \in [1..n], \forall d \in [0,d_i), (s_i, v_i + d) \models \varphi \\ \left(\sum_{i=1}^{n} d_i\right) \in I \ \textit{et} \ (s_n, v_n) \models \psi \end{cases} \\
(M,v) \models \forall\varphi\, \mathcal{U}_I \psi & \text{ssi} & \forall\sigma \in [\![\mathcal{N}]\!] \ \textit{nous avons} \ \begin{cases} (s_0,v_0) = (M,v) \\ \forall i \in [1..n], \forall d \in [0,d_i), (s_i, v_i + d) \models \varphi \\ \left(\sum_{i=1}^{n} d_i\right) \in I \ \textit{et} \ (s_n, v_n) \models \psi \end{cases}
\end{array}
$$

FIG. 6.1 – Semantique de *TPN-TCTL*

6.2 TCTL pour les réseaux de Petri temporels

6.2.1 *TPN-TCTL*

Nous introduisons un TCTL [ACD90] pour les TPN (*TPN-TCTL*) dont les propositions atomiques sont des contraintes d'exclusion mutuelle généralisées (Generalized Mutual Exclusion Constraints : *GMEC*) et nous donnons un résultats de décidabilité du model-checking d'une formule *TPN-TCTL* sur un TPN borné.

Nous étendons d'abord les contraintes d'exclusion mutuelle généralisées (*GMEC*) introduites dans [GDS92] de la manière suivante :

Définition 23 (GMEC) *Soit un TPN avec n places $P = \{p_1, p_2, \cdots, p_n\}$. Une GMEC est définie inductivement par :*

$$\mathit{GMEC} ::= \left(\sum_{i=1}^{n} a_i * \mathbf{M}(p_i)\right) \bowtie c \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \varphi \Rightarrow \psi$$

où $a_i \in \mathbb{Z}$, $\mathbf{M}$ est un mot-clé, $p_i \in P$, $\bowtie \in \{<, \leq, =, >, \geq\}$, $c \in \mathbb{N}$ et $\varphi, \psi \in$ GMEC. Les opérateurs ($\vee$,$\wedge$,$\Rightarrow$) ont la signification habituelle.

Intuitivement, la signification de $\mathbf{M}(p_i) \bowtie c$ est que le marquage courant de la place p_i est en relation $\bowtie$ avec c.

Nous donnons maintenant une définition de *TPN-TCTL* basée sur un *subscript*[4] TCTL [ACD90] et les GMEC.

Définition 24 (TCTL pour les TPN) *La logique temporelle TPN-TCTL est définie inductivement par :*

$$\mathit{TPN\text{-}TCTL} ::= \mathit{GMEC} \mid \mathbf{false} \mid \neg\varphi \mid \varphi \Rightarrow \psi \mid \exists\varphi\, \mathcal{U}_I \psi \mid \forall\varphi\, \mathcal{U}_I \psi \tag{6.1}$$

où **false** *est un mot-clé, $\varphi, \psi \in$ TPN-TCTL, $I \in \{[a,b], [a,b), (a,b], (a,b)\}$ avec $a \in \mathbb{N}$ et $b \in \mathbb{N} \cup \{\infty\}$.*

[4]Pour lequel les opérateurs sont étendus avec un intervalle spécifiant une contrainte de temps sur une séquence de tirs sans désigner explicitement une horloge.

Les opérateurs ($\vee$,$\wedge$,...) ont la signification classique et nous utilisons les raccourcis habituels $\mathbf{true} = \neg\mathbf{false}$, $\exists\Diamond_I\phi = \exists\mathbf{true}\,\mathcal{U}_I\phi$, $\forall\Diamond_I\phi = \forall\mathbf{true}\,\mathcal{U}_I\phi$, $\exists\Box_I\phi = \neg\forall\Diamond_I\neg\phi$, $\forall\Box_I\phi = \neg\exists\Diamond_I\neg\phi$.

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

Soit $\sigma = (s_0, v_0) \xrightarrow{(d_1,t_1)} \cdots \xrightarrow{(d_n,t_n)} (s_n, v_n) \in [\![\mathcal{N}]\!]$ une exécution de $\mathcal{N}$. La valeur de vérité d'une formule φ de *TPN-TCTL* pour un état (M, v) est donnée par la figure 6.1.

6.2.2 Décidabilité et complexité de TCTL pour les TPN

La traduction des TPN vers les TA que nous donnons dans le chapitre précédent complétée par [CR04] est déjà une preuve de la décidabilité de *TPN-TCTL* mais nous en donnons ici une preuve directe basée sur le graphe des régions de [AD94].

Définition 25 (Région) *Soit C l'ensemble des horloges d'un TPN ($x \in C$ est l'horloge associée à $t_x \in T$). Pour tout $x \in C$, soit $c_x = \beta_x$ si $\beta_x \neq \infty$ et $c_x = \alpha_x$ sinon. Pour $a \in \mathbb{R}_{\geq 0}$ nous notons $\lfloor a \rfloor$ la partie entière de a et $\langle a \rangle$ sa partie fractionnelle.*

La relation d'équivalence $\simeq$ est définie sur les valuations des horloges de C. $v \simeq v'$ ssi toutes les conditions suivantes sont vérifiées :

1. *Pour tout $x \in C$, soit $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, soit les deux valuations ($v(x)$ et $v'(x)$) sont plus grandes que c_x.*
2. *Pour tout $x, y \in C$ avec $v(x) \leq c_x$ et $v(y) \leq c_y$, $\langle v(x) \rangle \leq \langle v(y) \rangle$ ssi $\langle v'(x) \rangle \leq \langle v'(y) \rangle$.*
3. *Pour tout $x \in C$ avec $v(x) \leq c_x$, $\langle v(x) \rangle = 0$ ssi $\langle v'(x) \rangle = 0$.*

Une région *est une classe d'équivalence des valuations d'horloges induite par la relation $\simeq$.*

Définition 26 (Graphe des TPN-régions) *Étant donnés un TPN $\mathcal{N}$ et sa sémantique donnée par le système de transitions $\mathcal{S}_N$, le graphe des TPN-régions est le quotient de $\mathcal{S}_N$ par la relation d'équivalence $\simeq$.*

Le graphe des TPN-régions permet de prouver les théorèmes suivants sur les TPN. Les preuves sont similaires à celle de [AD94] et [ACD90] sur le graphe des régions classique.

Théorème 7 *Etant donné un TPN borné $\mathcal{N}$, le nombre de ses régions est borné par :*

$$|C|! \cdot 2^{|C|} \cdot \prod_{y \in C} (2 \cdot c_y + 2)$$

Théorème 8 (Décidabilité de *TPN-TCTL*) *TPN-TCTL est décidable pour les TPN bornés.*

Le calcul de ce graphe des régions est exponentiel en le nombre de transitions simultanément sensibilisées et linéaire en le nombre de marquages accessibles dans le TPN. Cependant d'après les résultats de [AD94] :

Théorème 9 (PSPACE) *Il existe un algorithme PSPACE* [5] *basé sur le graphe des TPN-régions pour décider TPN-TCTL sur les TPN.*

[5] En considérant que la taille d'un TPN est le nombre de ses marquages accessibles.

De plus, nous donnons ici la démarche de la preuve que le model-checking *TPN-TCTL* est au moins PSPACE-hard en considérant une définition classique de la taille d'un TPN (nombre de places + nombre de transitions).

Théorème 10 (PSPACE-hardness) *Le model-checking d'une formule TPN-TCTL sur un TPN borné est PSPACE-hard.*

Preuve. Nous prouvons que le problème de la décision de la valeur de vérité d'une formule booléenne quantifiée (Quantified Boolean Formula : QBF) peut être réduit à du model-checking de *TPN-TCTL*. Étant donné que ce problème est connu comme étant PSPACE-hard [HU79], nous en déduisons que *TPN-TCTL* pour les TPN bornés est PSPACE-hard. □
La preuve complète est dans [Gar05].

6.3 Vérification à la volée des TPN

Le graphe des régions de la section précédente n'est pas utilisable en pratique pour vérifier les propriétés de *TPN-TCTL* à cause du problème de l'explosion du nombre de régions. Nous allons donc nous intéresser à une méthode à la volée plus efficace basée sur une autre abstraction de l'espace d'états.

Un algorithme de model-checking à la volée génère l'espace d'états et vérifie la propriété concernée simultanément. Contrairement à l'approche consistant à générer l'ensemble de l'espace d'état avant la vérification, cela permet de stopper l'exploration dès que la valeur de vérité de la propriété est connue.

L'efficacité des méthodes à la volée a été prouvée sur les automates temporisés avec des outils tels qu'Uppaal. Ces outils restreignent l'ensemble des propriétés TCTL dans le but d'utiliser des algorithmes plus efficaces mais les cas d'étude ont montré que la classe de propriétés vérifiables était suffisante pour les cas pratiques étudiés.

Nous proposons donc un sous-ensemble de *TPN-TCTL* pour lequel nous pouvons faire efficacement de la vérification à la volée sur des TPN bornés.

6.3.1 *TPN-TCTL*$_{\mathcal{S}}$

Définition 27 (*TPN-TCTL*$_{\mathcal{S}}$) *TPN-TCTL*$_{\mathcal{S}}$ *est la sous-classe de la logique temporelle TPN-TCTL définie par :*

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

où $\varphi, \psi \in$ *GMEC,* $(\varphi \rightsquigarrow_I \psi) = \forall\Box(\varphi \Rightarrow \forall\Diamond_I\psi)$, $I \in \{[a,b],[a,b),(a,b],(a,b)\}$ *avec* $a \in \mathbb{N}$ *et* $b \in \mathbb{N} \cup \{\infty\}$, *et* $I_l \in \{[0,c],[0,c)\}$ *avec* $c \in \mathbb{N}$.

Précisons que *TPN-TCTL*$_{\mathcal{S}}$ est définie non-inductivement mais que $\exists\Box_I$, $\forall\Box_I$, $\exists\Diamond_I$, $\forall\Diamond_I$ et $\rightsquigarrow_{I_l}$ peuvent être définies inductivement en utilisant $\exists\mathcal{U}_I$ and $\forall\mathcal{U}_I$ et les opérateurs usuels.

La propriété de réponse bornée : $\varphi \rightsquigarrow_I \psi$ correspond à $\forall\Box(\varphi \Rightarrow \forall\Diamond_I\psi)$. Par exemple, la réponse bornée $\varphi \rightsquigarrow_{[0,c]} \psi$ exprime que lorsque la GMEC φ devient vraie, la GMEC ψ doit devenir vrai en au maximum c unités de temps.

6.3.2 Espace d'états abstrait basé sur les zones

Dans [GRR03], nous proposons le graphe des zones (défini à l'origine pour les TA) pour le calcul de l'espace d'états des TPN. Les essais effectués dans [GRR03] montrent que cette méthode est particulièrement efficace par rapport aux autres abstractions basées sur les classes d'états. Nous proposons dans cette section une amélioration de cette méthode permettant de réduire la taille du graphe généré et nous montrons comment l'appliquer à la vérification à la volée des propriétés *TPN-TCTL*$_{\mathcal{S}}$.

Nous donnons d'abord quelques définitions classiques.

Définition 28 (Zone) *Soit C un ensemble d'horloges. Une* zone *est un ensemble convexe de valuations d'horloges représentant un ensemble de contraintes de la forme $x_i - x_j \leq c_{ij}$, $x_i \leq c_{i0}$, $x_i \geq c_{0i}$ avec $c_{ij}, c_{i0}, c_{0j} \in \mathbb{Z}$.*

De même que pour les classes d'états de Berthomieu, une zone peut être encodée par une DBM (Difference Bound Matrices [BM83, Dil89]) en utilisant une horloge additionnelle x_0 toujours égale à 0. Nous associons à cette horloge une transition t_0 toujours sensibilisée mais jamais tirée.

Une zone Z représente l'ensemble des contraintes atomiques : $\forall x_i, x_j, (x_i - x_j \leq z_{ij})$. La forme canonique d'une DBM peut être obtenue par un calcul basé sur l'algorithme *Floyd-Warshall*. Dans la suite, nous ne considérons les DBM que sous leur forme canonique.

Définition 29 (État Symbolique) *Un* état symbolique *d'un TPN est un couple (M, Z) où M est un marquage et Z est une zone sur l'ensemble des horloges associées aux transitions sensibilisées par le marquage M.*

Informellement, un état symbolique représente un ensemble de valuations pour lesquelles un marquage est accessible.

Définition 30 (Successeurs Discrets) *Soit $s = (M, Z)$ un état symbolique. Les* successeurs discrets *de s obtenus par le tir de la transition t sont :*

$$Post_t(s) = (M', Z') = \begin{cases} M' = M - {}^\bullet t + t^\bullet \\ Z' = (Z \cap \{x_t \geq \alpha_t\})\,[X_e \leftarrow 0] \end{cases}$$

où X_e est l'ensemble des horloges nouvellement sensibilisées et $Z[X \leftarrow 0]$ représente « la zone obtenue à partir de Z en mettant à zéro les horloges de X ' ».

$Post_t(s)$ est l'ensemble des états qui sont accessibles à partir de s par le tir de la transition discrète t.

Définition 31 (Successeurs temporels) *Soit $s = (M, Z)$ un état symbolique. Les* successeurs temporels *de s sont les états de l'*état symbolique *:*

$$\overrightarrow{Post}(s) = (M, \overrightarrow{Z} \bigcap_{t \in Enabled(s)} \{x_t \leq \beta_t\})$$

$\overrightarrow{Post}(M, Z)$ est l'ensemble des états qui sont accessibles à partir de s en laissant s'écouler du temps (jusqu'à ce qu'une horloge atteigne la borne maximale de tir.

Nous notons $\overrightarrow{Post_t}(s) = \overrightarrow{Post}(Post_t(s))$,

Algorithme L'algorithme calcule itérativement, à partir d'un état symbolique, l'ensemble des états symboliques accessibles. A partir de l'état symbolique $s = (M, Z)$, une itération de l'algorithme est donnée par :

1. Calcul des états accessibles par écoulement du temps $(M, Z') = \overrightarrow{Post}(M, Z)$.
2. Pour chaque transition t tirable dans (M, Z'), calcul de tous les successeurs discrets $(M', Z'') = Post_t(M, Z')$.

Si le réseau comporte des transitions avec une borne de tir maximale infinie, une étape supplémentaire est nécessaire consistant à appliquer une approximation assurant la convergence. Dans le cas contraire, aucune approximation n'est nécessaire et l'algorithme calcule uniquement et exactement les états du TPN.

La présentation complète de l'algorithme ainsi que les preuves de sa correction et son exactitude relativement à l'accessibilité de marquage du TPN sont données dans [GRR03].

6.3.3 Amélioration de l'approximation

Comme nous l'avons mentionné précédemment, lorsqu'un TPN contient des bornes maximales infinies, une étape d'approximation est nécessaire pour obtenir une abstraction finie de son espace d'états. Cette opération est connue sous le nom de *normalisation* ou de *k-approximation*.

Dans le cas des TPN, cela consiste à choisir un entier k égal à la plus grande constante (finie) du TPN. Chaque état symbolique (M, Z) est alors approximé dans (M, Z') en utilisant la fonction *k-approx* suivante :

Définition 32 (k-*approx*) *Soit Z une zone (sous forme canonique) sur un ensemble d'horloges x_i. k-approx $(Z) = Z'$ est définie par :*

$$\forall t_i, t_j \in \mathit{Enabled}(M) \cup \{t_0\},\ \ z'_{ij} = \begin{cases} \infty & \text{si } z_{ij} > k\ ; \\ -k & \text{si } z_{ij} < -k\ ; \\ z_{ij} & \text{sinon.} \end{cases}$$

L'algorithme calcule à chaque itération : $\overrightarrow{Post}^k_t(s) = k\text{-}approx\left(\overrightarrow{Post}_t(s)\right)$

L'idée sous-jacente est que, si une horloge va au-delà de la valeur k, sa valeur précise n'a pas d'importance et les comportements résultant de la zone obtenue sont les mêmes qu'à partir de (M, Z). Nous étendons cette idée en proposant de prendre des valeurs de k différentes selon la transition non bornée considérée.

Définition 33 (k_x-*approx*) *Soit $s = (M, Z)$ un état symbolique. k_x-approx $(s) = s' = (M, Z')$ est défini par :*

$$\forall t_i, t_j \in \mathit{Enabled}(M) \cup \{t_0\},\ \ z'_{ij} = \begin{cases} \infty & \text{si } z_{ij} \geq \alpha_i \wedge \beta_i = \infty\ ; \\ -\alpha_j & \text{si } z_{ij} \leq -\alpha_j \wedge \beta_j = \infty\ ; \\ z_{ij} & \text{sinon.} \end{cases}$$

Par convention : $\alpha_0 = \beta_0 = 0$.

TPN	Nb. d'états symboliques		
	k-approx	k_x-*approx*	k'_x-*approx*
train2.xml	84	72	57
train3.xml	1030	813	546
train4.xml	53 882	16 702	7434
train5.xml	2 795 480	414 926	124 354
train6.xml	-	-	2 427 393
traffic-lights.xml	24 475	6 744	4877

Tab. 6.1 – Comparaison *k-approx* / k_x-*approx* / k'_x-*approx*

En appliquant l'algorithme *en avant* utilisant k_x-*approx* à la place de *k-approx*, nous construisons une sur-approximation de l'espace d'états. Cette sur-approximation conduit à une abstraction plus compacte mais restant exacte vis-à-vis de l'accessibilité de marquage et de la vérification des propriétés de *TPN-TCTL*$_{\mathcal{S}}$ (voir section 6.3.5, théorèmes 11 et 12).

De plus, cette k_x-*approximation* peut encore être améliorée en considérant la définition suivante :

Définition 34 (k'_x-*approx*) *Soit $s = (M, Z)$ un état symbolique. k'_x-approx$(s) = s' = (M, Z')$ est défini par :*

$$\forall t_i, t_j \in \mathit{Enabled}(M) \cup \{t_0\},\ \ z'_{ij} = \begin{cases} 0 & \text{si } (\beta_i = \infty \wedge i = 0); \\ \infty & \text{si } (i \neq j \wedge i \neq 0 \wedge (\beta_j \neq \infty \vee z_{ij} - \alpha_i \geq z_{0i})); \\ z_{ij} & \text{sinon.} \end{cases}$$

Nous comparons ces trois méthodes en termes de nombre d'états symboliques calculés pour représenter l'espace d'états de TPN avec des transitions non-bornées (avec une borne maximale infinie) et nous obtenons le tableau 6.1. Les exemples traités sont ceux du passage à niveau dont le modèle TPN est proposé dans [BV03] et décrit dans les figures 6.2 et 6.3.

L'utilisation de la k'_x-*approximation* est donc particulièrement intéressante puisqu'elle permet une réduction importante du nombre d'états symboliques calculés.

6.3.4 Model-checking à la volée de *TPN-TCTL*$_{\mathcal{S}}$

Proposition 1 *(Méthode pour le Model-checking de TPN-TCTL$_{\mathcal{S}}$)*

Puisque nous avons choisi d'ajouter un intervalle de temps aux opérateurs de TPN-TCTL$_{\mathcal{S}}$, nous avons besoin d'une horloge supplémentaire pour vérifier si une séquence du TPN vérifie la contrainte temporelle de la formule en cours de vérification. Cette horloge que nous appellerons c mesure la durée des séquences du TPN et son ajout est équivalent à l'ajout d'une transition (avec comme intervalle $[0, \infty[$) qui n'est jamais tirée.

Puisque cette horloge n'est pas bornée, nous devons lui choisir une constante k_c pour réaliser la k_x-approximation. Cette valeur est choisie ainsi :

- *si la formule TPN-TCTL$_{\mathcal{S}}$ a un intervalle borné $I = [c_{min}, c_{max}]$ nous prenons $k_c = c_{max} + 1$ ce qui assure le calcul de la durée exacte des séquences de tir du TPN.*
- *si la formule TPN-TCTL$_{\mathcal{S}}$ a un intervalle non borné $I = [c_{min}, \infty[$ nous prenons $k_c = c_{min} + 1$ ce qui assure que les séquences de durée supérieure à c_{min} sont détectables.*

Une preuve de cette proposition est présentée dans la section 6.3.5 avec le théorème 12.

En conséquence, nous pouvons construire à la volée l'espace d'états d'un TPN avec l'algorithme *en avant* présenté dans la section précédente.

Algorithme. Nous ne détaillerons pas ici l'ensemble des algorithmes pour la vérification de $TPN\text{-}TCTL_{\mathcal{S}}$ mais nous donnons le schéma de l'algorithme pour la vérification de $\forall\varphi U_I\psi$:

Soit $Z = (M, Z)$ un état symbolique pour lequel nous voulons tester la propriété $\forall\varphi U_I\psi$ (avec $I = [c_{min}, c_{max}]$). Nous utilisons une horloge supplémentaire c pour mémoriser la contrainte temporelle. Intuitivement, nous pouvons itérativement décider de la valeur de vérité de $\forall\varphi U_I\psi$ de la façon suivante :

- Décision :
  - S'il existe des valuations de l'horloge c plus grandes que c_{max} alors la propriété est fausse.
  - Si $M \models \psi$ et si toutes les valuations de l'horloge c de Z sont dans I alors la propriété est vraie pour cet état symbolique.
  - Si $M \not\models \varphi$ alors la propriété est fausse.
- Successeurs :
  - Nous calculons les successeurs temporels de Z.
  - Nous supprimons tous les états tels que φ est vrai et $c \in I$ et notons (M, Z') ce nouvel état symbolique (ces états sont supprimés car ils vérifient déjà tous la propriété).
  - Nous calculons les successeurs par le franchissement de transitions.
- Convergence :
  - L'algorithme converge par inclusion de zone. Si le réseau est Zénon, une technique de mémorisation de préfixe de la trace en cours d'analyse permet de détecter les boucles.

6.3.5 Correction de la méthode

Pour prouver la correction de cette méthode nous allons montrer que l'abstraction utilisée est exacte c'est à dire consistante[6] et complète[7].

Définition 35 *Une évolution d'un état q est une trace temporisée à partir de l'état q. Les évolutions d'un ensemble d'états sont l'union de toutes les évolutions de ses états.*

Nous allons donc montrer qu'un état symbolique (M, Z) et son approximation ont exactement les mêmes évolutions. Pour cela, nous montrons dans le théorème 11, qu'une zone Z et son approximation ont le même domaine de tir c'est-à-dire correspondent à la même classe d'états [BV03]. Nous rappelons d'abord comment on calcule le domaine de tir d'une zone.

Définition 36 ([BB93, BV03]) *Soit (M, Z) un état symbolique (Z est sous forme canonique) et (M, v) l'un de ses états. Le domaine de tir I_v de (M, v) est défini par $\forall t_i \in enabled(M), I_{v_i} = [max(0, \alpha_i - v_i), \beta_i - v_i]$.*
Le domaine de tir de (M, Z) est l'union des domaines de tir de tous ses états c'est à dire : $\{(M, I_v) | (M, v) \in (M, Z)\}$

La forme canonique d'un domaine de tir peut être calculée directement à partir de Z.

[6] Toute évolution du modèle est aussi dans le ZBG.

[7] Toute évolution dans le ZBG est aussi possible dans le modèle.

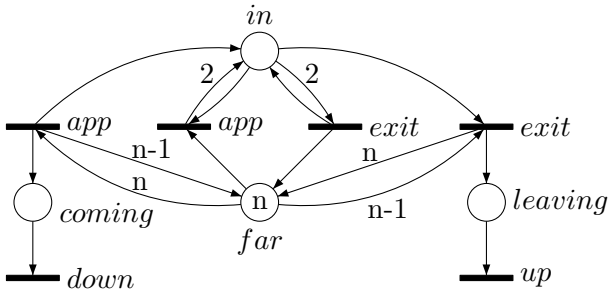


Fig. 6.2 – Modèle du contrôleur du passage à niveau

Théorème 11 *Soit (M, Z) un état symbolique sous forme canonique. (M, Z), sa k_x-approximation (M, Z') et sa k'_x-approximation (M, Z'') ont le même domaine de tir.*

Preuve. La preuve est dans [Gar05] □

Théorème 12 *La proposition 1 est exacte pour le model-checking TPN-TCTL$_\mathcal{S}$.*

Preuve. Sur un TPN $\mathcal{N}$, l'algorithme *en avant* basé sur les zones (sans k-approximation)[8] calcule exactement tous les états et les évolutions de la sémantique $\mathcal{S}_\mathcal{N}$ [GRR03]. De plus le théorème 11 prouve qu'un état symbolique et son approximation (k_x-*approx* ou k'_x-*approx*) ont le même domaine de tir. Dans [BD91] et [BB93], les auteurs ont montré que toutes les classes partageant le même marquage et le même domaine de tir ont également les mêmes évolutions. Par conséquent, notre abstraction est exacte : elle ajoute des états mais préserve l'accessibilité des marquages et les évolutions.

Pour vérifier *TPN-TCTL$_\mathcal{S}$*, nous ajoutons une horloge pour laquelle nous appliquons la « k_c-approximation » particulière de la proposition 1. La preuve du théorème 11 s'étend facilement pour toute approximation de valeur supérieure à celle proposée pour la k_x-approximation. En conséquence, toutes les évolutions relatives à cette horloge sont exactes. □

6.4 Exemple du passage à niveau

Un prototype du *model-checker* à la volée a été implémenté et intégré dans notre outil Roméo [GLMR05] d'édition et d'analyse des TPN. La version courante permet de vérifier les propriétés *TPN-TCTL$_\mathcal{S}$* sur un TPN borné. En fonction de la formule à vérifier, une trace (contre-exemple) peut être générée pour être analysée. Les approximations k-*approx*, k_x-*approx* et k'_x-*approx* sont implémentées.

Passage à niveau Considérons l'exemple classique du passage à niveau (*n trains, n voies et un passage à niveau*). La version TPN de cet exemple est proposée dans [BV03]. Le réseau est obtenu par la composition parallèle de n modèles du train (fig. 6.3) synchronisés avec le modèle du contrôleur du passage à niveau (*app*, *exit*, instanciés n fois) (fig. 6.2) et le modèle du passage à niveau (*down*, *up*) (fig. 6.3).

[8] Sans garantie de terminaison

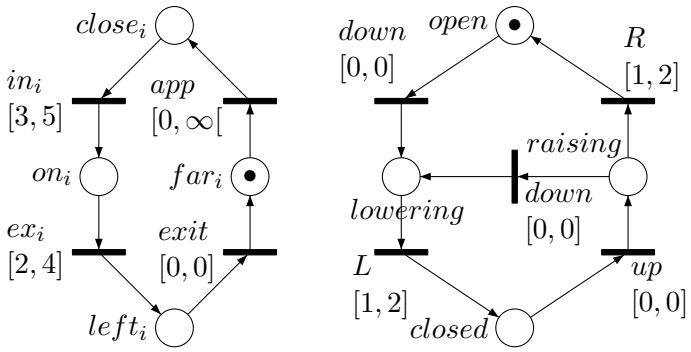


FIG. 6.3 – Modèles du train et du passage à niveau

Propriétés. Des propriétés non quantitatives telles que *lorsqu'il n'y a pas de train en approche, le passage à niveau s'ouvre fatalement* peuvent être vérifiées sur le graphe des classes atomiques [BV03] calculés par l'outil TINA.

Avec notre méthode et en utilisant notre implémentation dans ROMÉO, nous pouvons montrer que les propriétés suivantes sont vraies :

- $\phi_1 = \forall\Box\big(\vee_{i\in[1,n]} On_i\big) \Rightarrow Closed$ c.à.d. lorsqu'un train passe le passage à niveau, celui-ci est fermé ;
- $\phi_2 = Lowering \rightsquigarrow_{[0,2]} Closed$ c.à.d. le passage à niveau se ferme toujours en moins de 2 unités de temps ;
- $\phi_3 = far \rightsquigarrow_{[0,2]} (\neg far \vee open)$ c.à.d. lorsque aucun train ne s'approche, le passage à niveau est fermé ou sera fermé dans moins de 2 unités de temps (où far signifie $M(far) = n$).

Au delà des propriétés CTL telles que ϕ_1, l'intérêt principal de la méthode que nous venons de décrire est de permettre de vérifier des propriétés temporelles quantitatives telles que ϕ_2 et ϕ_3 sur les TPN.

6.5 Conclusion

Nous avons étudié le model-checking de TCTL sur les réseaux de Petri temporels. D'un point de vue théorique, nous avons prouvé en utilisant le graphe des régions la décidabilité du model-checking de *TPN-TCTL* sur les TPN bornés et avons montré que sa complexité est PSPACE. Cependant, d'un point de vue pratique, le graphe des régions n'est pas utilisable à cause du problème de l'explosion du nombre de région. Nous nous sommes donc intéressés aux méthodes à la volée dont l'efficacité a été prouvée sur les automates temporisés avec des outils tel qu'UPPAAL. Ces méthodes restreignent l'ensemble des propriétés TCTL dans le but d'utiliser des algorithmes plus efficace mais les cas d'étude ont montré que la classe de propriétés vérifiables était suffisante pour les cas pratiques étudiés. Nous avons donc considéré un sous ensemble de *TPN-TCTL* ($TPN\text{-}TCTL_{\mathcal{S}}$) pour lequel nous avons proposé un algorithme *en avant* de model-checking à la volée efficace utilisant une abstraction basée sur les zones. Comme pour les automates temporisés, un opérateur de surapproximation des zones est utilisé pour assurer la terminaison des algorithmes. Afin d'accélérer la convergence, nous avons défini une approximation plus fine des zones conduisant à une abstraction plus compacte de l'espace d'états. Nous avons montré que notre abstraction est exacte vis à vis des propriétés de $TPN\text{-}TCTL_{\mathcal{S}}$. La méthode est implémentée et intégrée à notre outil ROMÉO [GLMR05].

Il est ainsi possible de vérifier des propriétés temps réel sur les TPN avec cet outil et de bénéficier de ses autres fonctionnalités : génération d'un contre-exemple lorsque la propriété est fausse, environnement de simulation.

Chapitre 7

Vérification d'applications temps réel avec ordonnancement préemptif

But de l'étude. Les systèmes temps réel sont généralement composés de plusieurs tâches qui interagissent et partagent un ou plusieurs processeurs. Ainsi, pour un système S, les tâches doivent être ordonnancées sur les processeurs de façon à ce qu'elles respectent un certain nombre de propriétés P imposées notamment par l'application contrôlée. Cela est obtenu, en général, par l'utilisation soit d'une approche *hors-ligne*, soit d'une approche *en-ligne*. Dans l'approche hors-ligne, un échéancier est construit avant l'exécution du système de façon à ce que S vérifie les propriétés P. Dans l'approche en-ligne, l'ordonnancement des tâches est effectué lors de l'exécution selon une politique d'ordonnancement définie, généralement basée sur un système de priorités (par exemple *Rate Monotonic*, *Earliest Deadline First*). Il faut alors vérifier que le système S vérifie bien les propriétés P et en particulier qu'il est bien ordonnançable, c'est-à-dire que chaque tâche respecte son échéance temporelle.

Depuis les travaux fondateurs de Liu et Layland en 1973 [LL73], l'analyse de l'ordonnancement en ligne a été très étudiée et de nombreux résultats *analytiques* ont été proposés qui concernent principalement l'ordonnançabilité d'ensembles de tâches. Des algorithmes peu coûteux et exacts permettent de déterminer l'ordonnançabilité d'ensembles de tâches indépendantes avec des durées d'exécution fixes : par exemple [Tin94, PH98, HD03].

Des extensions ont été proposées qui permettent la prise en compte d'interactions entre les tâches et des durées d'exécution variables [PH99, HKL91]. Elles fournissent des bornes supérieures des temps de réponse des tâches et constituent donc des conditions suffisantes d'ordonnançabilité. Cela conduit à un pessimisme inhérent, qui augmente potentiellement avec la complexité du système considéré, et constitue donc une motivation pour l'utilisation de méthodes de vérification formelles avec des modèles tels que les automates temporisés [AD94] et les réseaux de Petri temporels [Mer74, BM83] qui permettent notamment la prise en compte de mécanismes de synchronisation complexes et de durées d'exécution des tâches variables.

Cependant les automates temporisés et les réseaux de Petri temporels ne sont en général pas suffisants pour modéliser et vérifier les applications temps réel en présence d'ordonnancement préemptif. En effet, dans ces modèles, le temps s'écoule de façon identique pour toutes les composantes du système, ce qui permet de modéliser des politiques d'ordonnancement non préemptives ou des tâches qui s'exécutent chacune sur un processeur différent. Par contre, ils ne permettent pas de représenter les politiques d'ordonnancement préemptives où l'exécution

d'une tâche est interrompue et reprise au même endroit un peu plus tard.

Dans ce chapitre, nous présentons une méthode de vérification de propriétés temporelles quantitatives pour les systèmes temps réel avec ordonnancement préemptif : le système, modélisé sous la forme d'un réseau de Petri temporel étendu à l'ordonnancement (*Scheduling*-TPN), est d'abord traduit en un automate à chronomètres qui lui est temporellement bisimilaire. La vérification de propriétés temporelles quantitatives est ensuite réalisée à l'aide de l'outil HyTech. L'efficacité de cette approche s'appuie sur deux points forts : premièrement, la traduction met en œuvre une minimisation du nombre de chronomètres de l'automate résultat, ce qui est un paramètre critique pour l'efficacité de la vérification qui s'ensuit. Deuxièmement, la traduction est effectuée par un algorithme surapproximant, à base de *Difference Bound Matrix*, donc efficace, mais qui produit tout de même un automate temporellement bisimilaire. La méthode a été implémentée et nous fournissons des résultats expérimentaux illustrant son efficacité.

Contexte des travaux. Nous avons débuté ces travaux avec Anne-Marie Deplanche en définissant dans [RD01, RD02] une extension des TPN (*Scheduling*-TPN) permettant de prendre en compte la façon dont les tâches réparties sur les différents processeurs sont ordonnancées et en proposant une surapproximation du calcul de l'espace d'état des *Scheduling*-TPN basée sur le graphe des classes d'état.

Ces travaux ont ensuite été poursuivis dans le cadre de la thèse de Didier Lime [Lim04]. Dans un premier temps [LR03a], nous avons étudié l'expressivité des *Scheduling*-TPN vis à vis des problèmes concrets d'application temps-réel et nous avons proposé une methode de calcul exacte de l'espace d'état des *Scheduling*-TPN. Dans un deuxième temps, en collaboration avec Bernard Berthomieu et Francois Vernadat, nous avons étudié la décidabilité des problèmes tels que l'accessibilité, la k-bornitude et la vivacité des *Scheduling*-TPN [BLRV05, BLRV04]. Enfin, nous avons proposé dans [LR04, LR05b] une traduction des *Scheduling*-TPN vers les automates à stopwatch qui constitue le coeur de ce chapitre.

Tout recemment, dans le cadre du DEA et de la thèse de Morgan Magnin, une méthode efficace de calcul de l'espace d'état des *Scheduling*-TPN a été proposée dans [MLR05]. Morgan Magnin poursuit actuellement les travaux sur ce modèle en considérant un temps discret.

7.1 Introduction

Modèles formels pour l'ordonnancement

Les modèles temporisés tels que les automates temporisés [AD94, HNSY94] ou les réseaux de Petri temporels [Mer74, BD91] ne sont en général pas suffisants pour modéliser et vérifier les applications temps réel en présence d'ordonnancement préemptif. Ils doivent pour cela être étendus afin de pouvoir modéliser une tâche interrompue et reprise au même endroit un peu plus tard.

Automates et ordonnancement

Les automates hybrides étendent les automates classiques à l'aide de variables continues dont les dynamiques d'évolution sont spécifiées dans chaque localité à l'aide d'une équation différentielle. Les automates hybrides linéaires forment une sous-classe des automates hybrides. Dans ce modèle, l'équation différentielle régissant l'évolution du vecteur des variables

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

Les automates à chronomètres [CL00] peuvent être définis comme des automates temporisés pour lesquels les horloges peuvent être arrêtées et redémarrées plus tard avec la même valeur. Ces horloges sont appelées chronomètres (*stopwatches*). Il s'agit donc d'une sous-classe *syntaxique* des automates hybrides linéaires. Cassez et Larsen montrent dans [CL00] que les automates à chronomètres avec des délais inobservables sont aussi expressifs que les automates hybrides linéaires [ACH+95b], dans le sens où, pour tout langage acceptable par un automate hybride linéaire, il existe un automate à chronomètres qui accepte le même langage. Le problème de l'accessibilité étant indécidable pour les automates hybrides linéaires, ils en déduisent qu'il l'est également pour les automates à chronomètres. Les auteurs proposent également une surapproximation de l'espace accessible à l'aide de matrices de différences (*Difference Bound Matrix* ou simplement DBM) [BM83, Dil89].

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

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

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

Réseaux de Petri temporels et ordonnancement

Un certain nombre d'auteurs proposent d'étendre les réseaux de Petri temporels afin de prendre en compte les aspects liés à l'interruption et à la reprise d'actions.

Okawa et Yoneda [OY96] proposent une extension dans laquelle les transitions sont groupées et où chaque groupe possède une vitesse d'exécution. Ces groupes correspondent à des tran-

sitions modélisant des activités concurrentes et qui sont prêtes à être tirées simultanément. Dans ce cas, leur vitesse d'exécution est divisée par la somme des vitesses d'exécution.

Dans [RD02], nous avons étendu les réseaux de Petri temporels pour prendre en compte la façon dont les tâches réparties sur les différents processeurs sont ordonnancées (*Scheduling*-TPN). Nous avons proposé le calcul d'une surapproximation de l'espace d'états basé sur le graphe de classes d'états classique de [BD91] et les DBMs.

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

Enfin, un modèle plus général de réseaux de Petri mettant en œuvre des chronomètres a été proposé dans [RL04] en étendant les réseaux de Petri temporels avec des arcs et hyperarcs inhibiteurs. Pour ces réseaux, le problème du calcul de l'espace d'états est le même que pour les réseaux de Petri temporels étendus de [RD02] et [BFSV04]. Pour ces trois modèles, nous avons démontré que la bornitude et l'accessibilité sont indécidables (même pour des réseaux bornés) [BLRV04, BLRV05]. Ils sont donc obtenus en explorant l'espace d'états à l'aide de semi-algorithmes.

7.2 Réseaux de Petri temporels étendus à l'ordonnancement

Les réseaux de Petri temporels étendus à l'ordonnancement (*Scheduling*-TPN, *Scheduling Time Petri Nets*) étendent les réseaux de Petri temporels en incluant dans la sémantique du modèle le comportement des ordonnanceurs temps réel. Cela implique la suspension et la reprise de l'exécution des tâches, lors des préemptions. Le temps s'arrêtant pour les tâches préemptées, ce modèle s'appuie sur le concept de chronomètre (ou *stopwatch*), horloge pour laquelle le temps peut être arrêté et redémarré.

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

L'ordonnanceur est donc ici considéré comme un contrôleur pour lequel le retour d'état ne dépend que du marquage courant avec une loi de contrôle déterminée par la fonction Act (figure 7.1).

Les *Scheduling*-TPN sont particulièrement bien adaptés à la modélisation des systèmes temps réel concrets. Le lecteur pourra se référer à [LR03a, Lim04] pour des détails à ce sujet.

7.2.1 Définitions

Définition 37 (*Scheduling*-TPN) *Un réseau de Petri temporel étendu à l'ordonnancement est un 8-uplet* $\mathcal{N} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0, Act)$*, où*

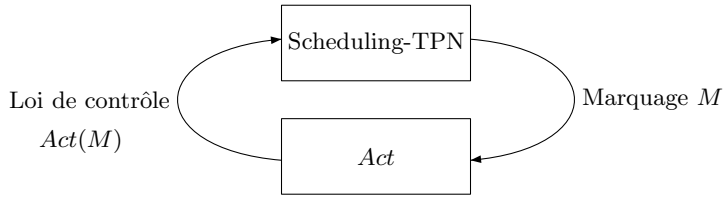


FIG. 7.1 – Contrôle (ordonnancement) par retour d'état

- $P = \{p_1, p_2, \ldots, p_m\}$ *est un ensemble fini et non vide de* places *;*
- $T = \{t_1, t_2, \ldots, t_n\}$ *est un ensemble fini et non vide de* transitions $(T \cap P = \emptyset)$ *;*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *est la* fonction d'incidence amont *;*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *est la* fonction d'incidence avale *;*
- $M_0 \in \mathbb{N}^P$ *est le* marquage initial *du réseau ;*
- $\alpha \in (\mathbb{R}^+)^T$ *et* $\beta \in (\mathbb{R}^+ \cup \{\infty\})^T$ *sont les fonctions donnant pour chaque transition, respectivement son instant de tir* au plus tôt *et* au plus tard *$(\alpha \leq \beta)$.*
- $Act \in (\mathbb{N}^P)^{\mathbb{N}^P}$ *est la* fonction de marquage actif *telle que* $\forall M \in \mathbb{N}^P, \forall p \in P, Act(M)(p) = M(p)$ *ou* $Act(M)(p) = 0$.

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

Une transition t est dite *sensibilisée* par le marquage M si $M \geq {}^\bullet t$[1], c'est-à-dire si le nombre de jetons, pour M, de chaque place amont de t est plus grand ou égal à la valuation de l'arc entre cette place et la transition. Nous notons $t \in enabled(M)$.

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

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

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

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

En plus des notions précédentes définies sur le réseau de Petri temporel classique sous-jacent, une transition t est dite *active* pour le marquage M, si elle est sensibilisée par le marquage $Act(M)$, c'est-à-dire $t \in enabled(Act(M))$.

Nous définissons la sémantique des réseaux de Petri temporels étendus à l'ordonnancement sous la forme d'un système de transitions temporisé (TTS) [LPY95].

Définition 38 (Sémantique d'un *Scheduling*-TPN) *La sémantique d'un réseau de Petri temporel étendu à l'ordonnancement $\mathcal{N}$ est définie sous la forme d'un système de transitions temporisé $S_\mathcal{N} = (Q, q_0, \Sigma, \rightarrow)$ tel que :*

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$ *;*

[1] les relations d'ordre sur les vecteurs sont considérées coordonnée par coordonnée.

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

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

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

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

7.2.2 Marquage actif

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

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

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

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

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

Pour pouvoir calculer le marquage actif, nous imposons la restriction que toutes les places amont d'une même transition doivent avoir la même priorité (ou pour certaines, ne pas être associées à un processeur : $\gamma(p) = \phi$). Sans cette restriction, nous pourrions avoir des relations

de priorité circulaires empêchant le calcul du marquage actif. En pratique, la possibilité d'avoir plusieurs places amont d'une transition affectée à des processeurs est rarement utile : elle permet de modéliser la dépendance d'une action à plusieurs ressources préemptibles. Dans la plupart des cas, il est préférable de respecter la règle suivante : toute transition doit avoir au plus une place amont p affectée à un « vrai » processeur, c'est-à-dire telle que $\gamma(p) \neq \phi$. Cela représente le fait que toute communication se fait par l'intermédiaire d'un service de l'exécutif.

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

Exemple

Considérons l'exemple de *Scheduling*-TPN présenté sur la figure 7.2 pour le cas particulier d'un ordonnancement à priorités fixes concernant les places p_3 et p_7. Ces deux places sont associées au même processeur $\gamma = 1$. La place p_7 a une priorité supérieure à celle de p_3. Lorsque ces deux places sont marquées, c'est la place p_7 qui sera marquée active $M(p_3) = M(p_7) = 1, Act(M)(p_3) = 0$ et $Act(M)(p_7) = 1$.

7.2.3 Décidabilité des *Scheduling*-TPN

Les problèmes de l'accessibilité d'un marquage ou d'un état, et le caractère borné sont indécidables pour les TPN [JLL77]. Il s'ensuit que ces problèmes sont également indécidables pour les *Scheduling*-TPN.

Bien qu'indécidables dans le cas général, ces problèmes deviennent décidables pour les TPN bornés. En effet, pour les TPN bornés, l'accessibilité de marquage peut être décidée en utilisant le graphe des classes d'états de [BM83], et l'accessibilité d'état ainsi que la vivacité sont décidées par les constructions alternatives de [BV03].

Nous avons prouvé dans [BLRV04, Lim04, BLRV05] que l'accessibilité d'état pour les TPN à chronomètres en général et les *Scheduling*-TPN en particulier, peut être réduite au problème de l'arrêt d'une machine à deux compteurs. L'accessibilité d'état est par conséquent indécidable pour les *Scheduling*-TPN, même bornés. Il suit que l'accessibilité d'un marquage, le caractère k-borné, et la vivacité, sont indécidables pour les *Scheduling*-TPN bornés.

7.2.4 Graphe des classes d'états

Comme pour les TPN, l'espace d'états d'un *Scheduling*-TPN est infini. En plus du fait que le réseau n'est pas nécessairement borné, le temps est dense (les horloges peuvent prendre un nombre infini de valeur). Il est donc nécessaire de calculer une abstraction de l'espace d'états. Cependant, étant donné que l'accessibilité est indécidable pour les *Scheduling*-TPN (même bornés), il n'existe pas nécessairement d'abstraction finie de l'espace d'états d'un *Scheduling*-TPN. Nous allons présenter dans cette section un semi-algorithme de calcul d'une abstraction exacte de l'espace basée sur les classes d'états de [BD91] puis une surapproximation utilisant des DBMs.

Classes d'états

La définition des classes d'états ne change pas par rapport à celle de [BD91] pour les réseaux de Petri temporels. La forme générale des domaines de tirs est néanmoins différente :

Définition 39 (Classe d'états) *Une* classe d'états C*, d'un réseau de Petri temporel étendu à l'ordonnancement, est un couple* (M, D) *où* M *est un marquage du réseau et* D *un polyèdre appelé* domaine de tir*.*

Le polyèdre D *prend la forme générale suivante :*

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

θ_i *représente les dates de tir possibles de la transition sensibilisée* t_i *depuis la classe.*

Soit D un polyèdre, nous noterons $[\![D]\!]$ l'ensemble de ses solutions.

Successeurs d'une classe d'états

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

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

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

- le nouveau marquage est calculé de façon classique par $M' = M - {}^\bullet t_f + t_f{}^\bullet$;
- le nouveau domaine de tir, D', est calculé selon les étapes suivantes. Nous le noterons $next(D, t_f)$:
  1. changements de variables $\forall j,$ t.q. t_j est active, $\theta_j = \theta_f + \theta'_j$;
  2. $\forall j \neq t_f$, ajout des contraintes $\theta'_j \geq 0$;
  3. élimination des variables correspondant à des transitions désensibilisées par le tir de t_f (ce qui inclut donc t_f), par exemple en utilisant la méthode de Fourier-Motzkin [Dan63] ;
  4. ajout des inéquations relatives aux transitions nouvellement sensibilisées par le tir de t_f :
$$\forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq \theta'_k \leq \beta(t_k).$$

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

Cette différence avec les réseaux de Petri temporels classiques n'est cependant pas anodine, puisqu'elle entraîne la génération de domaines de tirs qui ne peuvent pas être représentés par des DBMs mais par un polyèdre de forme générale [RL04, LR03a].

Graphe des classes d'états

Pour un *Scheduling*-TPN $\mathcal{N}$, avec la forme particulière des classes d'états, la nouvelle règle de tirabilité des transitions et le nouveau calcul des successeurs, nous définissons le système de transitions $\Gamma'(\mathcal{N}) = (\mathbf{C}, C_0, \Sigma, \rightarrow)$ où :

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

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

Nous définissons l'égalité de deux classes d'états comme suit :

Définition 41 (Egalité de classes d'états) *Deux classes d'états $C = (M, D)$ et $C' = (M', D')$ d'un réseau de Petri temporel étendu à l'ordonnancement sont dites* égales *si $M = M'$ et $[\![D]\!] = [\![D']\!]$. Nous notons $C \equiv C'$.*

En pratique, l'égalité est testée sur les formes canoniques des domaines de tir. En effet, à un choix de base de sous-espace vectoriel près, nous pouvons déterminer une forme canonique des polyèdres permettant de les comparer syntaxiquement.

On définit alors le graphe des classes $\Gamma(\mathcal{N})$ du réseau de Petri temporel étendu à l'ordonnancement $\mathcal{N}$ par le quotient de $\Gamma'(\mathcal{N})$ par la relation d'équivalence $\equiv$: $\Gamma(\mathcal{N}) = \Gamma'(\mathcal{N})/\equiv$.

Surapproximation

Pour les réseaux de Petri temporels classiques, les domaines de tir des classes d'états ne comportent que des inéquations du types $\theta_i - \theta_j \leq \gamma_{ij}$ et $\alpha_i \leq \theta_i \leq \beta_i$. Ces types bien particuliers peuvent être encodés et manipulés efficacement à l'aide de matrices de différences (DBMs).

En raison de la forme générale des domaines de tir des classes d'états des *Scheduling*-TPN, le calcul de ces classes implique la manipulation de polyèdres généraux ce qui est bien plus coûteux que la manipulation des DBMs

Par conséquent, nous avons proposé, dans [LR03a], une surapproximation naturelle des classes d'états qui consiste à supprimer du domaine toutes les inéquations ne pouvant pas être représentées par des DBMs.

Puisqu'il s'agit d'une surapproximation, cela présente le risque de permettre le tir de transitions qui ne le sont normalement pas. Cela est illustré par le réseau de la figure 7.2. Après le tir de la séquence t_4, t_1, t_5, la transition t_6 n'est pas tirable car soit t_2 soit t_3 *doit* être tirée avant elle, en fonction de l'instant de tir de t_4. La classe obtenue est :

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

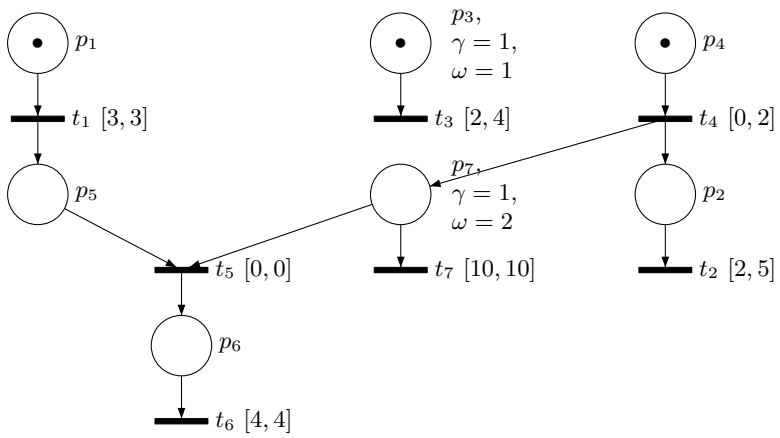


Fig. 7.2 – Un réseau de Petri temporel étendu à l'ordonnancement

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

En ajoutant des comportements, cette approximation présente le risque de rendre non borné le marquage des classes d'états et donc leur nombre. D'un autre côté, elle peut permettre au calcul des classes de se terminer en « coupant » des contraintes temporelles qui le ferait diverger sinon. Enfin, excepté le risque d'être pessimiste, la surapproximation n'est pas gênante pour la vérification de propriétés de sûreté.

7.3 Automate à chronomètres des classes d'états

Dans cette section, nous montrons comment traduire les réseaux de Petri temporels étendus à l'ordonnancement *bornés* en automates à chronomètres. Ces automates à chronomètres pourront ensuite être vérifiés à l'aide de l'outil HyTech [HHWT97].

Plusieurs arguments plaident en faveur de cette technique par rapport à une modélisation directe du système en automates à chronomètres : certains systèmes temps réel s'expriment de façon plus naturelle et - ou - de façon plus concise (au prix d'une plus grande spécificité) sous la forme d'un réseau de Petri temporel étendu à l'ordonnancement que sous la forme d'un automate à chronomètres. En particulier, l'ordonnanceur doit être modélisé explicitement pour les automates à chronomètres alors qu'il est implicite avec les réseaux de Petri temporels étendus à l'ordonnancement car son comportement est inclus dans la sémantique du modèle. Par ailleurs, la méthode permet une modélisation utilisant conjointement les deux modèles. Cependant, nous disposons avec cette méthode d'un argument plus fort. En effet, une modélisation générique d'un système temps réel directement sous la forme d'automates à chronomètres requiert typiquement de faire le produit d'un automate par tâche ainsi qu'un automate par ordonnanceur (donc par processeur) ce qui nous donne au moins autant de chronomètres. Or, lorsque l'opération converge (le problème est indécidable), le coût de la vérification d'un automate hybride est exponentiel en le nombre de chronomètres

en conséquence de quoi, lorsque ce nombre augmente, le calcul de l'espace d'états devient rapidement infaisable en pratique avec HyTech.

La traduction que nous proposons d'un réseau de Petri temporel étendu à l'ordonnancement borné en automate à chronomètres est, comme dans [LR03b], conçue de façon à réduire le plus possible le nombre de chronomètres de l'automate résultat. Cette réduction est d'autant plus nécessaire que les méthodes syntaxiques de réduction d'horloges du type [DY96] sur les automates temporisés n'ont à ce jour pas été étendues aux automates à chronomètres.

Enfin, et surtout, nous pouvons faire cette traduction en utilisant la surapproximation présentée dans la section précédente et obtenir quand même un automate à chronomètres bisimilaire temporellement au réseau de Petri temporel étendu à l'ordonnancement initial. En effet, les localités ajoutées par la surapproximation ne sont en fait pas accessibles dans l'automate résultat.

En résumé, cette méthode propose, à partir d'un réseau de Petri temporel étendu à l'ordonnancement, obtenu par exemple en utilisant les motifs génériques proposés dans [LR03a], de calculer rapidement (car le calcul est basé sur des DBMs) un[2] automate à chronomètres temporellement bisimilaire possédant moins de chronomètres qu'une modélisation directe à l'aide d'automates à chronomètres génériques. Cela permet ainsi d'augmenter de façon importante la taille des systèmes vérifiables en pratique par HyTech.

7.3.1 Automates à chronomètres

Les automates à chronomètres (*stopwatch automata* ou SWA) peuvent être vus comme une extension des automates temporisés à l'aide de chronomètres, c'est-à-dire d'horloges pour lesquelles l'écoulement du temps peut être suspendu puis repris plus tard. Par exemple, considérons l'automate à chronomètre de la figure 7.4. Dans la localité C_0 le chronomètre x_2 est actif ; le temps s'écoule pour lui. Dans la localité C_1 le chronomètre x_2 est arrêté.

Une *contrainte atomique* sur une variable x est une formule de la forme $x \bowtie c$ pour $c \in \mathbb{Q}_{\geq 0}$ et $\bowtie \in \{<, \leq, \geq, >\}$. Nous notons $\mathcal{C}(X)$ l'ensemble des *contraintes* obtenues par conjonction de contraintes atomiques sur l'ensemble des variables X.

Définition 42 (Automate à chronomètres) *Un* automate à chronomètres *est un 7-uplet* $(L, l_0, X, A, E, Inv, Dif)$ *où :*

- L *est un ensemble fini de* localités *;*
- l_0 *est la* localité initiale *;*
- X *est un ensemble fini de* chronomètres *à valeurs réelles positives ;*
- A *est un ensemble fini d'*actions *;*
- $E \subset L \times \mathcal{C}(X) \times A \times 2^X \times 2^{X^2} \times L$ *est un ensemble fini d'*arêtes. *Soit* $e = (l, \delta, \alpha, R, \rho, l') \in E$. *e est l'arête reliant la localité* l *à la localité* l'*, avec la* garde δ*, l'action* α*, l'ensemble de chronomètres à remettre à zéro* R *et la fonction d'affectation de chronomètres* $\rho : X \to X$.
- $Inv \in \mathcal{C}(X)^L$ *associe un* invariant *à chaque localité ;*
- $Dif \in (\{0,1\}^X)^L$ *associe à chaque localité l'*activité *des chronomètres dans cette localité.* $\dot{X}$ *désigne l'ensemble des dérivées par rapport au temps des variables de* X *:* $\dot{X} = (Dif(l)x)_{x \in X}$.

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

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

Soit ν une valuation, R un sous-ensemble de X et $\rho : X \to X$. Alors $\nu' = \nu[R \leftarrow 0][\rho]$ est la valuation telle que : $\forall x \in R, \nu'(x) = 0$ et $\forall x \notin R, \nu'(\rho(x)) = \nu(x)$.

Définition 43 (Sémantique d'un SWA) *La sémantique d'un automate à chronomètres $\mathcal{A}$ est définie sous la forme d'un système de transitions temporisé $\mathcal{S}_{\mathcal{A}} = (Q, Q_0, \Sigma, \to)$ où*

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

$$(l,\nu) \xrightarrow{a} (l',\nu') \text{ ssi } \exists (l,\delta,a,R,\rho,l') \in E \text{ tel que } \begin{cases} \delta(\nu) = \texttt{true}, \\ \nu' = \nu[R \leftarrow 0][\rho], \\ Inv(l')(\nu') = \texttt{true} \end{cases}$$

  - *la relation de transition* continue *:*

$$(l,\nu) \xrightarrow{d} (l,\nu') \text{ ssi } \begin{cases} \nu' = \nu + \dot{X} * d, \\ \forall d' \in [0,d], Inv(l)(\nu + \dot{X} * d') = \texttt{true} \end{cases}$$

7.3.2 Automate à chronomètres des classes d'états

Nous allons maintenant montrer comment traduire un réseau de Petri temporel borné en un automate à chronomètres, par le calcul de son espace d'états lorsque celui-ci converge[3]. Par ailleurs, la méthode que nous proposons met l'accent sur la minimisation du nombre de chronomètres de l'automate à chronomètres obtenu.

Classes d'états étendues

La construction des classes d'états de la section précédente est basée sur un temps relatif : dans le domaine de tir d'une classe d'états, on contraint les instants de tir des transitions sensibilisées exprimés par rapport à la date d'entrée dans la classe. Les valuations ν des états de la sémantique du *Scheduling*-TPN (définition 38) sont « oubliées ». Nous allons les rendre à nouveau explicites à l'aide des chronomètres de l'automate que nous allons construire.

Intuitivement, nous pouvons voir que pour, générer cet automate à chronomètres, il serait suffisant d'associer un chronomètre à chaque transition du réseau de Petri initial. Cependant, dans un souci d'optimisation du nombre de chronomètres, nous allons déterminer pour chaque classe d'états calculée le nombre de chronomètres nécessaires pour exprimer les valuations des transitions sensibilisées. Nous définissons donc des *classes d'états étendues* de la façon suivante :

Définition 44 (Classe d'états étendues) *Une classe d'états étendue, d'un réseau de Petri temporel étendu à l'ordonnancement, est un 4-uplet $(M, D, \chi, trans)$, où M est un marquage, D un polyèdre appelé* domaine de tir*, χ un ensemble de chronomètres et $trans \in (2^T)^\chi$ associe à chaque chronomètre un ensemble de transitions.*

[3] L'accessibilité est indécidable pour les *Scheduling*-TPN.

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

Successeur d'une classes d'états étendue

Le calcul des successeurs diffère de celui du graphe des classes d'états par les calculs requis pour les deux composantes supplémentaires. D'une part, nous avons une création dynamique des chronomètres en fonction des besoins. D'autre part, il est nécessaire de calculer l'activité de tous les chronomètres car celle-ci est susceptible de changer en fonction de l'activité des transitions. De plus, il faut également assurer la cohérence de l'association entre les transitions et les chronomètres qui les représentent ; un chronomètre actif (respectivement inactif) peut représenter uniquement les valuations de transitions actives (respectivement inactives). Nous avons donc les étapes supplémentaires suivantes :

1. pour chaque chronomètre x de χ, les transitions désensibilisées par le tir de t_f sont retirées de $trans(x)$;
2. les chronomètres dont l'image par $trans$ est vide sont retirés de χ ;
3. s'il y a des transitions nouvellement sensibilisées par le tir de t_f, deux cas sont possibles :
   - il existe un chronomètre x dont la valeur est 0. Alors, les transitions nouvellement sensibilisées sont ajoutées à $trans(x)$,
   - il n'existe pas de chronomètre nul. Alors nous créons un nouveau chronomètre x_i associée aux transitions nouvellement sensibilisées. L'indice i choisi est le plus petit indice disponible parmi ceux des chronomètres de χ. Nous ajoutons x_i à χ et $trans(x_i)$ est l'ensemble des transitions nouvellement sensibilisées.
4. si toutes les transitions associées à un chronomètre actif (respectivement inactif) x sont inactives (respectivement actives) alors celui-ci est stoppé : $\dot{x} = 0$ (respectivement activé : $\dot{x} = 1$) ;
5. si l'image par $trans$ d'un chronomètre actif (respectivement inactif) x contient à la fois des transitions actives et des transitions inactives, alors un nouveau chronomètre x' est créé, de la même façon que pour les transitions nouvellement sensibilisées, auquel sont associées par $trans$ les transitions inactives (respectivement actives). Ce chronomètre est inactif (respectivement actif) et il prend la valeur de x : $x' = x$.

Graphe des classes d'états étendues

Le calcul de l'automate à chronomètres des classes d'états se fait en deux étapes. Dans la première, nous allons calculer un graphe des classes d'états étendues de façon assez similaire au graphe des classes d'états classiques. Le graphe obtenu est cependant structurellement différent du graphe classique. En effet, comme nous le verrons, le critère de convergence est différent.

À partir de ce graphe des classes étendues, nous déduirons syntaxiquement l'automate à chronomètres de ce graphe.

La classe initiale est définie par le marquage initial, le domaine $\{\alpha(t_i) \leq \theta_i \leq \beta(t_i)\}$, et les chronomètres x_0 et x_1 associées par $trans$ respectivement aux transitions actives et non

actives. Bien sûr, s'il n'y a pas de transition non active, la classe initiale ne comporte qu'un seul chronomètre : x_0.

Nous pouvons définir, avec la classe initiale et la relation de transition définie ci-dessus, le graphe des classes d'états étendues $\Delta'(\mathcal{N})$ du réseau de Petri temporel étendu à l'ordonnancement $\mathcal{N}$. Nous définissons d'abord le système de transitions $\Delta''(\mathcal{N}) = (C^{ext}, C_0, \Sigma, \rightarrow^{ext})$:

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

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

Définition 45 (Stopwatch-similarité) *Deux classes d'états étendues $C = (M, D, \chi, trans)$ et $C' = (M', D', \chi', trans')$ sont* stopwatch-similaires*, ce que nous notons $C \approx C'$, si elles ont le même marquage, le même nombre de chronomètres et si leurs chronomètres sont associés aux mêmes transitions :*

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

D'un point de vue théorique, nous pouvons définir le graphe des classes d'états étendues $\Delta'(\mathcal{N})$ d'un réseau de Petri temporel étendu à l'ordonnancement $\mathcal{N}$ comme le quotient de $\Delta''(\mathcal{N})$ par la relation de stopwatch-similarité : $\Delta'(\mathcal{N}) = \Delta''(\mathcal{N})/\approx$. Notamment, dans ce quotient, un nombre (potentiellement) infini de classes sont regroupées, ainsi qu'un nombre (potentiellement) infini de transitions avec la même étiquette, la même source et la même cible.

D'un point de vue pratique, nous ne pouvons pas calculer Δ''. Par conséquent, nous convergeons dès que la classe nouvellement calculée $C' = (M', D')$ est *stopwatch*-similaire à une classe précédemment calculée $C = (M, D)$. Puis nous poursuivons l'exploration pour les successeurs de la classe $(M, D' \setminus D)$. Quitte à recalculer une partie des successeurs[4], cela revient à continuer le calcul des successeurs lorsque l'inclusion de classes d'états étendues définie ci-après n'est pas satisfaite :

Définition 46 (Inclusion de classes d'états étendues) *Une classe d'états étendue $C' = (M', D', \chi', trans')$ est* incluse *dans une classe d'états étendue $C = (M, D, \chi, trans)$ si C et C' sont stopwatch-similaires et $[\![D']\!] \subset [\![D]\!]$. Nous notons $C' \subset C$.*

Automate à chronomètres des classes d'états

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

Définition 47 (Automate à chronomètres des classes d'états) *L'*automate à chronomètres des classes d'états $\Delta(\mathcal{T}) = (L, l_0, X, A, E, Inv)$ *est défini syntaxiquement à partir du graphe des classes d'états étendues par :*

[4]Il serait bien moins efficace de calculer effectivement la soustraction.

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

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

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

Les localités de l'automate sont les classes d'états étendues. Pour chacune d'elle, l'invariant est obtenu à partir des bornes supérieures des intervalles de tir des transitions sensibilisées par le marquage. La garde de chaque transition sortante correspond à la borne inférieure de l'intervalle de tir de la transition du réseau correspondante. L'ensemble des chronomètres remis à zéro est soit vide, soit le singleton constitué de l'horloge associée aux transitions nouvellement sensibilisées dans la classe obtenue par le tir de la transition. Enfin les affectations de chronomètres proviennent soit de la création de nouveaux chronomètres pour maintenir la cohérence entre l'activité des transitions et celles des chronomètres qui les représentent, soit d'un renommage dans le cas d'une convergence avec une classe *stopwatch*-similaire.

7.3.3 Exemple

La figure 7.3 présente un réseau de Petri temporel étendu à l'ordonnancement modélisant deux tâches périodiques s'exécutant en concurrence sur le même processeur et synchronisées par un sémaphore. La tâche 1 ($task1$ à gauche) et la tâche 2 ($task2$ à droite) sont toutes les deux composés de 2 modules séquentiels $M1$ et $M2$. Les durées d'exécution (sans compter les préemptions) des modules sont respectivement dans les intervalles [1,4] et [2,4] pour la tâche 1 et [2,3] et [3,5] pour la tâche 2. La période des 2 tâches est de 20 unités de temps et la requête de la tâche 1 subit un offset de 1 unité de temps. Enfin la tâche 2 incrémente un sémaphore à la fin de son module $M1$ et la tâche 1 décrémente ce même sémaphore à la fin de son module $M2$. La figure 7.4 montre l'automate à chronomètres des classes d'états correspondant.

7.3.4 Propriétés

Bisimulation

Lorsque le calcul se termine, l'automate à chronomètres produit et le réseau de Petri temporel étendu à l'ordonnancement initial sont temporellement bisimilaires, ce qui prouve la correction de la traduction.

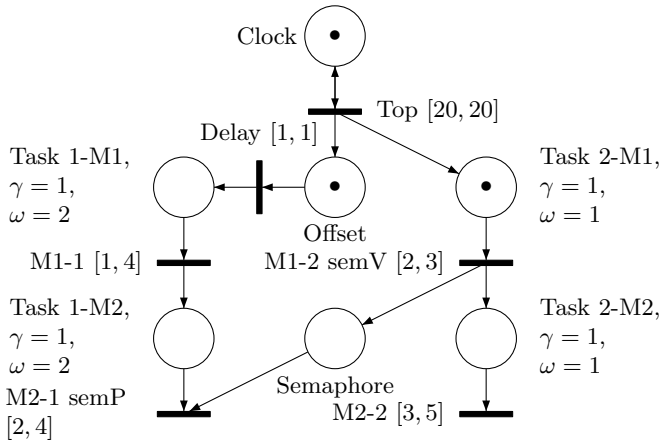


Fig. 7.3 – *Scheduling*-TPN modélisant deux tâches sur un même processeur synchronisées par un sémaphore.

Théorème 13 (Bisimulation) *Soient un Scheduling-TPN $\mathcal{N}$ et $\mathcal{A} = (L, l_0, X, A, E, Inv, Dif)$, son automate à chronomètres des classes d'états défini dans la section 7.3.2. Soient $Q_{\mathcal{N}}$ l'ensemble des états du réseau $\mathcal{N}$ et $Q_{\mathcal{A}}$ l'ensemble des états de $\mathcal{A}$. Soit $\mathcal{R} \subset Q_{\mathcal{N}} \times Q_{\mathcal{A}}$ une relation binaire telle que $\forall s = (M_{\mathcal{N}}, \nu_{\mathcal{N}}) \in Q_{\mathcal{N}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}$,*

$$s\mathcal{R}a \Leftrightarrow \begin{cases} M_{\mathcal{N}} = M_{\mathcal{A}}{}^{5} \\ \forall t \in enabled(M_{\mathcal{N}}), \exists x \in X \text{ t.q. } \nu_{\mathcal{N}}(t) = \nu_{\mathcal{A}}(x) \text{ et } \begin{cases} \dot{x} = 1 \text{ si } t \text{ est active} \\ \dot{x} = 0 \text{ sinon} \end{cases} \\ \forall x \in X, \exists t \in enabled(M_{\mathcal{N}}) \text{ t.q. } \nu_{\mathcal{N}}(t) = \nu_{\mathcal{A}}(x) \text{ et } t \text{ est active ssi } \dot{x} = 1 \end{cases}$$

$\mathcal{R}$ est une bisimulation.

Preuve. La preuve est dans l'annexe A.4. □

Surapproximation

Nous avons montré dans la section précédente que le domaine des classes d'états peut être surapproximé par une DBM. Nous pouvons bien sûr faire la même chose pour les classes d'états étendues. Nous obtenons alors un automate à chronomètres dont le comportement contient celui du réseau de Petri temporel étendu à l'ordonnancement. En fait nous allons montrer que cet automate est toujours temporellement bisimilaire au réseau de Petri. En effet, puisque les gardes et invariants sont calculés de manière syntaxique, les états supplémentaires ajoutés par la surapproximation ne sont pas accessibles :

Théorème 14 (Bisimulation) *Soit $Q_{\mathcal{N}}$ l'ensemble des états du réseau de Petri temporel $\mathcal{N}$ et $Q_{\mathcal{A}}$ l'ensemble des états de son automate à chronomètres des classes d'états* surapproximé *$\mathcal{A} = (L, l_0, X, A, E, Inv, Dif)$. Soit $\mathcal{R} \subset Q_{\mathcal{N}} \times Q_{\mathcal{A}}$ une relation binaire telle que $\forall s = (M_{\mathcal{N}}, \nu_{\mathcal{N}}) \in Q_{\mathcal{N}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}$,*

$$s\mathcal{R}a \Leftrightarrow \begin{cases} M_{\mathcal{N}} = M_{\mathcal{A}} \\ \forall t \in enabled(M_{\mathcal{N}}), \exists x \in X \text{ t.q. } \nu_{\mathcal{N}}(t) = \nu_{\mathcal{A}}(x) \text{ et } \begin{cases} \dot{x} = 1 \text{ si } t \text{ est active} \\ \dot{x} = 0 \text{ sinon} \end{cases} \\ \forall x \in X, \exists t \in enabled(M_{\mathcal{N}}) \text{ t.q. } \nu_{\mathcal{N}}(t) = \nu_{\mathcal{A}}(x) \text{ et } t \text{ est active ssi } \dot{x} = 1 \end{cases}$$

$\mathcal{R}$ est une bisimulation.

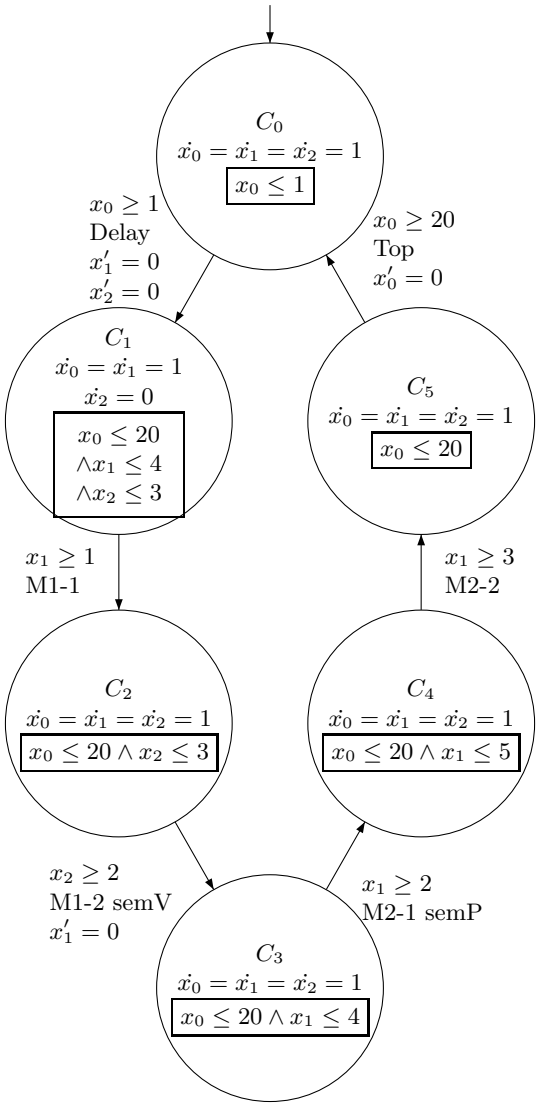


FIG. 7.4 – Automate à chronomètres des classes d'états du réseau de la figure 7.3. La localité initiale est C_0.

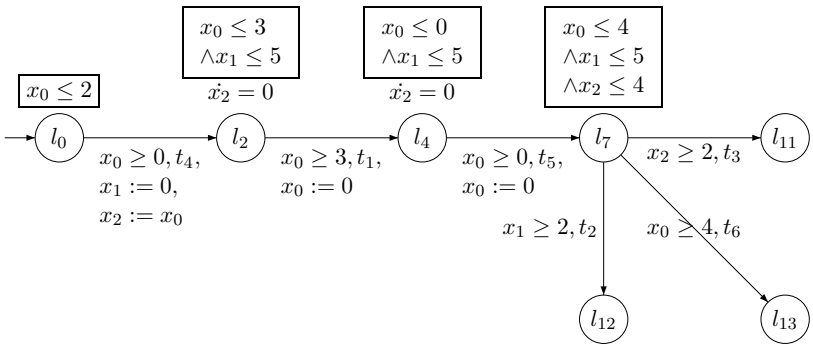


Fig. 7.5 – Traduction partielle en SWA du réseau de la figure 7.2

La bisimulation est la même que pour l'automate exact et la preuve également. Pour s'en convaincre complètement, supposons qu'il existe dans la localité l une arête sortante $e = (l, \delta, t, R, \rho, l')$ car t est tirable dans la classe d'états *surapproximée* l alors qu'elle ne l'est pas dans la classe *exacte* correspondante. Si nous supposons qu'avant d'atteindre la localité l, le comportement de l'automate était correct (exact), alors à l'instant précis de l'entrée dans la classe l, l'automate est dans un état $a = (M, \nu_{\mathcal{A}})$ qui est en relation avec un état $s = (M, \nu_{\mathcal{N}})$ du *Scheduling*-TPN $\mathcal{R}$. D'une part, puisque t est en fait non tirable, cela signifie qu'il existe une autre transition t' qui doit être tirée avant elle : $\alpha(t) - \nu_{\mathcal{N}}(t) > \beta(t') - \nu_{\mathcal{N}}(t')$. Donc, par définition de $\mathcal{R}$, il existe un chronomètre x_t tel que $\alpha(t) - \nu_{\mathcal{A}}(x_t) > \beta(t') - \nu_{\mathcal{N}}(t')$. Puisque, par définition d'un *Scheduling*-TPN, $\beta(t') \geq \nu_{\mathcal{N}}(t')$, cela nous donne $\alpha(t) - \nu_{\mathcal{A}}(x_t) > 0$.

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

Cela est illustré par la figure 7.5 qui présente la partie de l'automate à chronomètres des classes d'états du réseau de la figure 7.2 de la section précédente correspondant à la séquence à laquelle nous nous étions intéressés. Nous voyons que la localité l_{13} correspondant au tir « parasite » de la transition t_6 n'est en fait pas accessible. En effet, à l'entrée dans la localité l_7, nous avons les équations suivantes $x_0 = 0$, $x_1 = 3 - \tau$ et $x_2 = \tau$, en notant τ l'instant de tir de t_4 ($0 \leq \tau \leq 2$). Par conséquent, nous pouvons laisser s'écouler $\min\{4, 2 + \tau, 4 - \tau\}$ unités de temps et il est clair que $\forall \tau \in]0, 2], 4 - \tau < 4$ et $\forall \tau \in [0, 2[, 2 + \tau < 4$. Donc la garde $x_0 \geq 4$ ne sera jamais vérifiée.

En conclusion, nous voyons que le calcul à l'aide d'une surapproximation peut ajouter des comportements au graphe des classes d'états étendues. Cependant, ces comportements indésirables sont éliminés par l'ajout syntaxique des gardes et invariants pour obtenir l'automate à chronomètres final. Le comportement de ce dernier est donc *exactement* le même (au sens de la bisimulation temporelle) que celui du *Scheduling*-TPN initial.

7.4 Vérification

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

Nous avons comparé l'efficacité de notre méthode, implémentée dans l'outil Roméo [GLMR05], avec une modélisation générique directe en automate à chronomètres. Cette modélisation présenteé dans [Lim04] requiert un automate par tâche et un par ordonnanceur (et donc par processeur). Le résultat est donc un produit synchronisé d'automates à la Arnold-Nivat [Arn94].

Bien qu'elle ne soit pas présentée ici, la modélisation utilisant les *Scheduling*-TPN est toute aussi générique. On pourra se référer à [LR03a] pour des détails à ce sujet.

7.4.1 Résultats

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

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

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

Tab. 7.1 – Résultats expérimentaux

Pour comparer l'efficacité des deux approches, nous pouvons comparer la somme des deux dernières colonnes avec le nombre de la colonne 6. Nous constatons que, très vite, le calcul

de l'espace d'états du modèle issu de la modélisation générique directe sous la forme d'un produit d'automates à chronomètres devient impossible alors que la complexité du système augmente. Notre méthode, par contre, donne de bons résultats jusqu'à ce que l'automate qu'elle produit ne soit plus analysable non plus par HyTech (dernière ligne). Dans ce cas, pour des propriétés de sûreté, nous pouvons toujours utiliser le graphe des classes étendues généré par notre semi-algorithme à base de DBMs, en gardant à l'esprit que nous sommes alors en présence d'une surapproximation de l'espace d'états.

7.5 Conclusion

Nous avons présenté une méthode pour l'analyse efficace des systèmes temps réel en présence d'ordonnancement préemptif. Elle consiste dans un premier temps à modéliser le système sous la forme d'un réseau de Petri temporel étendu à l'ordonnancement. Dans un second temps, nous appliquons un semi-algorithme de traduction en automate à chronomètres. Ce semi-algorithme présente deux avantages principaux. Premièrement, il est rapide car il calcule des surapproximations de chaque classe d'états sous la forme de DBMs. L'automate à chronomètres résultat est néanmoins temporellement bisimilaire au réseau de Petri temporel étendu à l'ordonnancement initial car les localités supplémentaires de l'automate à chronomètres, générées éventuellement par l'approximation ne sont pas accessibles. Deuxièmement, il fournit un automate dont le nombre de chronomètres est minimisé par des techniques de réduction à la volée. La vérification de cet automate est donc plus efficace. Nous avons montré que, dans la pratique, cette approche est plus efficace qu'une modélisation générique directe du système sous la forme d'un produit synchronisé d'automates à chronomètres. Elle permet notamment de vérifier des systèmes bien plus importants en termes de nombre de processeurs et de tâches.

Les développements futurs comprennent la généralisation de la méthode à d'autres modèles à base de chronomètres (des automates à chronomètres vers les automates à chronomètres par exemple) et également à d'autres politiques d'ordonnancement que par priorités fixes. Enfin, nous souhaitons étudier la possibilité de synthétiser des contrôleurs, sous la forme d'un marquage actif $Act(M, \nu)$, basé non plus uniquement sur le marquage courant mais également les valuations des transitions.

Troisième partie

Conclusion

Chapitre 8

Conclusion et perspectives

Bilan

A l'heure du bilan que représente la rédaction d'une HDR, je crois que ma contribution dans le domaine de la vérification des systèmes temps réel est avant tout d'avoir établi des liens formels entre les réseaux de Petri temporels et les automates temporisés (et leurs extensions à chronomètres) ainsi qu'entre les méthodes de vérification qui leur sont associées. Ceci est illustré par la sélection de mes travaux présentée de manière détaillée dans la deuxième partie de ce mémoire.

En effet, nous avons, dans un premier temps, comparé l'expressivité des réseaux de Petri temporels et des automates temporisés. Nous avons montré que les automates temporisés étaient strictement plus expressifs que les réseaux de Petri temporels bornés en terme de bisimulation temporelle mais que ces deux modèles avaient la même expressivité en terme de langage temporisé. Ces résultats s'étendent très facilement aux extensions à chronomètres de ces deux modèles.

Dans un deuxième temps, nous nous sommes concentrés sur la vérification des réseaux de Petri temporels en nous inspirant des techniques existantes sur les automates temporisés. Nous avons ainsi proposé un TCTL pour les TPN : *TPN-TCTL*. Nous avons prouvé, en utilisant le graphe des régions, la décidabilité du model checking de *TPN-TCTL* sur les TPN bornés et avons montré que sa complexité est PSPACE. Nous nous sommes ensuite intéressés aux méthodes à la volée dont l'efficacité a été prouvée sur les automates temporisés avec des outils tel qu'Uppaal. Ces méthodes restreignent l'ensemble des propriétés TCTL dans le but d'utiliser des algorithmes plus efficaces mais les cas d'étude ont montré que la classe de propriétés vérifiables était suffisante pour les cas pratiques étudiés. Nous avons donc considéré un sous-ensemble de *TPN-TCTL* ($TPN\text{-}TCTL_{\mathcal{S}}$) pour lequel nous avons proposé un algorithme de *model-checking à la volée* efficace utilisant une abstraction basée sur les zones. Nous avons proposé une abstraction plus compacte de l'espace d'états et avons montré qu'elle est exacte vis à vis des propriétés de $TPN\text{-}TCTL_{\mathcal{S}}$. La méthode est implémentée et intégrée à notre outil Roméo [GLMR05]. Il est ainsi possible de vérifier des propriétés temps-réel sur les TPN avec cet outil et de bénéficier de ces autres fonctionnalités : génération d'un contre-exemple lorsque la propriété est fausse, environnement de simulation.

Enfin, dans un troisième temps, nous nous sommes intéressé aux modèles à chronomètres en proposant une méthode (basée à la fois sur les *Scheduling*-TPN et les automates à chro-

nomètres) de vérification d'applications temps réel avec ordonnancement préemptif. Elle consiste, dans une première étape, à modéliser le système sous la forme d'un réseau de Petri temporel étendu à l'ordonnancement. La deuxième étape consiste à appliquer un semi-algorithme de traduction en automate à chronomètres. Ce semi-algorithme est efficace car il calcule des surapproximations de chaque classe d'états sous la forme de DBMs ; nous avons prouvé que malgré l'utilisation de cette surapproximation dans le calcul, l'automate à chronomètres obtenu est temporellement bisimilaire au *Scheduling*-TPN initial. De plus, ce semi-algorithme fournit un automate dont le nombre de chronomètres (qui est un paramètre critique pour la vérification des automates à chronomètres) est minimisé par des techniques de réduction à la volée. Ainsi, cette méthode est, dans la pratique, beaucoup plus efficace qu'une modélisation générique directe du système sous la forme d'un produit d'automates à chronomètres. Elle permet notamment de vérifier des systèmes bien plus importants en termes de nombre de processeurs et de tâches. La méthode est implémentée et intégrée à notre outil Roméo et peut être appliquée en utilisant conjointement l'outil HyTech.

De plus, comme cela a été présenté dans la première partie de ce mémoire, je travaille également sur le problème du contrôle (et de la synthèse de contrôleur) dans les systèmes temporisés. En particulier, sur les réseaux de Petri temporels, nous avons étudié l'existence et la synthèse d'un contrôleur qui k-borne un réseau de manière à ce qu'il borne par exemple les éléments d'une application tels qu'un sémaphore, un tampon ou une pile. Enfin, nous avons obtenu très récemment des résultats sur le contrôle de la non-interférence temporisée qui offrent à notre avis de nombreuses perpectives.

Perspectives

Expressivité. Il existe plusieurs extensions temporelles des réseaux de Petri pour lesquelles le temps peut être associé aux transitions, aux places, aux arcs ou aux jetons. Seuls quelques travaux comparent ces différentes sémantiques et les résultats sont contradictoires. En nous aidant des résultats obtenus sur l'expressivité des réseaux de Petri T-temporels (où le temps est associé aux transitions) et des automates temporisés, nous pouvons certainement contribuer à comparer ces différentes extensions temporelles des réseaux de Petri en termes de langage et de bisimulation et affiner ainsi la classification sur les modèles temporisés.

Model-checking. Concernant le model-checking des réseaux de Petri termporels, nous avons seulement trouvé un encadrement de la complexité de *TPN-TCTL* qui reste donc un problème ouvert. De plus, la méthode à-la-volée que nous avons proposée ne peut s'appliquer qu'à $\mathit{TPN\text{-}TCTL}_{\mathcal{S}}$ (donc sans imbrication des formules) et il serait donc maintenant intéressant de proposer une méthode qui ne serait pas à-la-volée (donc moins efficace) mais sur l'ensemble des propriétés *TPN-TCTL*.

Contrôle de la non-interférence. Récemment, nous avons posé avec John Mullins et Guillaume Gardey les bases concernant la non-interférence dans le contexte temporisé. Cela ouvre un grand nombre de problèmes, particulièrement sur le contrôle de la non-interférence. En effet, les problèmes de décidabilité du contrôle pour plusieurs classes de non-interférence sont ouverts. J'ai l'intention de m'y consacrer pour tenter de les résoudre, de trouver les

méthodes et algorithmes permettant de décider de l'existence de ces contrôleurs puis de les synthétiser et enfin d'en trouver une implémentation.

Modèles à chronomètres. Les méthodes que nous avons proposées sur les modèles à chronomètres peuvent être généralisées à d'autres modèles et également à d'autres politiques d'ordonnancement (que celles par priorités fixes). De plus, nous n'avons pas abordé le problème du contrôle sur ces modèles et nous souhaitons étudier la possibilité de synthétiser des contrôleurs sous la forme d'une fonction $Act(M, \nu)$ basée non plus uniquement sur le marquage courant mais également les valuations des transitions. Enfin, nous avons commencé, dans le cadre de la thèse de Morgan Magnin, à étudier les approches impliquant une sémantique à temps discret qui paraissent prometteuses grâce aux progrès réalisés dans le calcul et la réprésentation symbolique d'espaces d'états de très grande taille pour les réseaux de Petri non temporisés [MDR02, Cia04].

Valorisation de l'outil Roméo. Notre logiciel Roméo de vérification des TPN ayant atteint un certain degré de maturité, j'ai été solicité à plusieurs reprises pour établir des liens entre des modèles de haut niveau décrits par des meta-langages (UML, ADL,CORE) et les réseaux de Petri. Il s'agit de définir une sémantique comportementale (temporelle) de ces modèles et de proposer une traduction vers les TPN préservant cette sémantique afin de pouvoir les simuler et les vérifier à partir de notre outil Roméo. Des travaux sur ce sujet sont en cours avec *Dassault Aviation* et d'autres devraient commencés avec la société *Sodius* sous la forme d'une Cifre en 2006.

Bibliographie

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for real-time systems. In *5th IEEE Symposium on Logic in Computer*, pages 414–425. IEEE Computer Society Press, june 1990.

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1) :2–34, may 1993.

[ACH+95a] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138 :3–34, 1995.

[ACH+95b] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138 :3–34, january 1995.

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

[AGS02] K. Altisen, G. Gössler, and J. Sifakis. Scheduler modelling based on the controller synthesis paradigm. *Journal of Real-Time Systems*, 23 :55–84, 2002. Special issue on control-theoritical approaches to real-time computing.

[AHH96] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22 :181–201, 1996.

[AlP+99] K. Altisen, G. Gößler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *20th IEEE Real-Time Systems Symposium (RTSS'99)*, pages 154–163, Phoenix, Arizona, USA, december 1999.

[AlS00] K. Altisen, G. Gößler, and J. Sifakis. A methodology for the construction of scheduled systems. In *FTRTFT'00*, volume 1926 of *Lecture Notes in Computer Science*, pages 106–120, Pune, India, september 2000. Springer-Verlag.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474, 1998.

[Arn94] A. Arnold. *Finite Transition System*. Prentice Hall, 1994.

[BB93] H. Boucheneb and G. Berthelot. Toward a simplified building of time Petri nets reachability graph. In *PNPM'93*, pages 46–55, Toulouse, France, october 1993.

[BCH+05a] Beatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier (H.) Roux. Comparison of different semantics for time Petri nets. In *Automated Technology for Verification and Analysis (ATVA'05)*, Lecture Notes in Computer Science, Taiwan, October 2005. Springer.

[BCH+05b] Beatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier (H.) Roux. Comparison of the expressiveness of timed automata and time Petri nets. In *3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2005)*, Lecture Notes in Computer Science, Uppsala, Sweden, sep 2005. Springer.

[BCH+05c] Beatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier (H.) Roux. When are timed automata weakly timed bisimilar to time petri nets ? In *25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2005)*, volume 3821 of *Lecture Notes in Computer Science*, Hyderabad, India, December 2005. Springer.

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE transactions on software engineering*, 17(3) :259–273, 1991.

[BD99] Marc Boyer and Michel Diaz. Non equivalence between time petri nets and time stream petri nets. In *Proceedings of 8th International Workshop on Petri Nets and Performance Modeling (PNPM'99)*, Zaragoza, Spain, 1999.

[Ber01] B. Berthomieu. La méthode des classes d'états pour l'analyse des réseaux temporels. In *Modélisation des Systèmes Réactifs (MSR'01)*, pages 275–290, Toulouse, France, october 2001. Hermes.

[BFSV03] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Modeling flexible real time systems with preemptive time Petri nets. In *15th Euromicro Conference on Real-Time Systems (ECRTS'2003)*, pages 279–286, 2003.

[BFSV04] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Time state space analysis of real-time preemptive systems. *IEEE transactions on software engineering*, 30(2) :97–111, February 2004.

[BH04] H. Boucheneb and R. Hadjidj. Towards optimal CTL* model checking of time Petri nets. In *WODES'04*, REIMS (France), June 2004.

[BLRV04] B. Berthomieu, D. Lime, O.H. Roux, and F. Vernadat. Reachability problems and abstract state spaces for time Petri nets with stopwatches. Technical Report 04483, Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS), Toulouse, France, October 2004.

[BLRV05] Bernard Berthomieu, Didier Lime, Olivier (H.) Roux, and Francois Vernadat. Problèmes d'accessibilité et espaces d'états abstraits des réseaux de Petri temporels à chronomètres. In *Modélisation des Systèmes Réactifs, (MSR'05)*, Grenoble, France, October 2005.

[BM83] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In *IFIP Congress Series*, volume 9, pages 41–46. Elsevier Science Publ. Comp. (North Holland), 1983.

[Bou02] Patricia Bouyer. Timed automata may cause some troubles. Technical report, LSV, July 2002.

[Bou03] Patricia Bouyer. Unteamable timed automata ! In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2003)*, volume 2607 of *LNCS*, pages 620–631, Berlin, Germany, February 2003. Springer Verlag.

[BST98] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. *Lecture Notes in Computer Science*, 1536 :103–129, 1998.

[BV03] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *TACAS'03*, pages 442–457. Springer–Verlag, Apr 2003.

[CEP00] L. A. Cortes, P. Eles, and Z. Peng. Verification of embedded systems using a petri net based representation. In *13th International Symposium on System Synthesis (ISSS'2000)*, 2000.

[Cia04] G. Ciardo. Reachability set generation for petri nets : Can brute force be smart ? In *ICATPN'04*, volume 3099 of *Lecture Notes in Computer Science*, pages 17–34, Bologna, Italy, june 2004. Springer-Verlag.

[CL00] Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In Catuscia Palamidesi, editor, *CONCUR'00*, number 1877 in Lecture Notes in Computer Science, pages 138–152, University Park, P.A., USA, July 2000.

[CR03] F. Cassez and O.H. Roux. Traduction structurelle des réseaux de Petri temporels vers les automates temporisés. In *Modélisation des Systèmes Réactifs, (MSR'03)*, Metz, France, october 2003.

[CR04] F. Cassez and O.H. Roux. Structural translation from time petri nets to timed automata. In *The 4th International Workshop on Automated Verification of Critical Systems (AVoCS 2004)*, London, United Kingdom, september 2004.

[Dan63] G.B. Dantzig. Linear programming and extensions. *IEICE Transactions on Information and Systems*, 1963.

[dFRA00] D. de Frutos Escrig, V. Valero Ruiz, and O. Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *ICATPN'00*, volume 1825 of *Lecture Notes in Computer Science*, pages 187–206, Aarhus, Denmark, june 2000.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Workshop Automatic Verification Methods for Finite-State Systems*, volume 407, pages 197–212, 1989.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *1996 IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81, Washington, DC, USA, december 1996.

[FMPY03] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis using two clocks. In Hubert Garavel and John Hatcliff, editors, *TACAS'03*, volume 2619 of *Lecture Notes in Computer Science*, pages 224–239, April 2003.

[FPY02] E. Fersman, P. Petterson, and W. Yi. Timed automata with asynchronous processes : Schedulability and decidability. In *TACAS'02*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82, Grenoble, France, 2002.

[Gar05] Guillaume Gardey. *Vérification et contrôle des réseaux de Petri temporels.* Thèse de doctorat, IRCCyN, December 2005.

[GDS92] A Giua, F. DiCesare, and M Silva. Generalized mutual exclusion constraints on nets with uncontrollable transitions. In *IEEE Int. Conf. on SMC*, 1992.

[GLMR05] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier (H.) Roux. Roméo : A tool for analyzing time Petri nets. In *17th International Conference on Computer Aided Verification (CAV'05)*, Lecture Notes in Computer Science, Edinburgh, Scotland, UK, July 2005. Springer.

[GMR05] Guillaume Gardey, John Mullins, and Olivier (H.) Roux. Non-interference control synthesis for security timed automata. In *3rd International Workshop on Security Issues in Concurrency (SecCo'05)*, ENTCS, San Francisco, USA, August 2005. Elsevier.

[GRR03] G. Gardey, O.H. Roux, and O.F. Roux. A zone-based method for computing the state space of a time Petri net. In *In Formal Modeling and Analysis of Timed Systems, (FORMATS'03)*, volume LNCS 2791. Springer-Verlag, September 2003.

[GRR06] G. Gardey, O.H. Roux, and O.F. Roux. State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems*, 2006. to appear.

[HD03] P.-E. Hladik and A.-M. Déplanche. Analyse d'ordonnan cabilité de tâches temps-réel avec offset et gigue. In *11th international Conference on Real-Time Systems (RTS'03)*, Paris, France, april 2003.

[HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech : A model checker for hybrid systems. *Journal of Software Tools for Technology Transfer*, 1(1-2) :110–122, 1997.

[HKL91] M.G. Harbour, M.H. Klein, and J.P. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *12th IEEE Real-Time Systems Symposium (RTSS'91)*, pages 116–128, San Antonio, USA, december 1991.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata ? In *Proceedings of the 27th Annual Symposium on Theory of Computing (STOC), ACM Press*, pages 373–382, 1995.

[HKPV98] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata ? *Journal of Computer and System Sciences*, 57 :94–124, 1998.

[HKSLT02] S. Haar, L. Kaiser, F. Simonot-Lion, and J. Toussaint. Equivalence of timed state machines and safe time petri nets. In *Proceedings of WODES 2002*, pages 119–126, 2002.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2) :193–244, 1994.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[JLL77] N.D. Jones, L.H. Landweber, and Y.E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science 4*, pages 277–299, 1977.

[KDC96] W. Khansa, J.-P. Denat, and S. Collart-Dutilleul. P-Time Petri Nets for manufacturing systems. In *International Workshop on Discrete Event Systems, WODES'96*, pages 94–102, Edinburgh (U.K.), august 1996.

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE transactions on software engineering*, 3(2) :125–143, 1977.

[Lim04] Didier Lime. *Vérification d'applications temps réel à l'aide de réseaux de Petri temporels étendus*. Thèse de doctorat, IRCCyN, December 2004.

[LL73] C. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, 20(1) :44–61, january 1973.

[LPY95] K.G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Fundamentals of Computation Theory*, pages 62–88, 1995.

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2) :134–152, october 1997.

[LR03a] D. Lime and O.H. Roux. Expressiveness and analysis of scheduling extended time Petri nets. In *5th IFAC International Conference on Fieldbus Systems and their Applications, (FET 2003)*, Aveiro, Portugal, July 2003. Elsevier Science.

[LR03b] D. Lime and O.H. Roux. State class timed automaton of a time Petri net. In *10th International Workshop on Petri Nets and Performance Models, (PNPM 2003)*. IEEE Computer Society, September 2003.

[LR04] D. Lime and O.H. Roux. A translation-based method for the timed analysis of scheduling extended time Petri nets. In *25th IEEE Real-Time Systems Symposium (RTSS 2004)*, Lisbon, Portugal, December 2004.

[LR05a] Didier Lime and Olivier (H.) Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Events Dynamic Systems - Theory and Applications (DEDS)*, 2005. to appear.

[LR05b] Didier Lime and Olivier (H.) Roux. Vérification formelle des systèmes temps réel avec ordonnancement préemptif. *Technique et Science Informatiques*, 2005. to appear.

[MD78] A.K. Mok and M. Dertouzos. Multiprocessor scheduling in a hard real-time environment. In *Seventh Texas Conf. on Computing Systems*, 1978.

[MDR02] P. Molinaro, D. Delfieu, and O.H. Roux. Improving the calculus of the marking graph of Petri nets with bdd-like structures. In *IEEE international conference on systems, man and cybernetics (SMC 2002)*, Hammamet, Tunisia, october 2002.

[Mer74] P.M. Merlin. *A study of the recoverability of computing systems.* PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1974.

[Min61] M. Minsky. Recursive unsolvability of Post's problem. *Ann. of Math*, 74 :437–454, 1961.

[MLR05] Morgan Magnin, Didier Lime, and 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)*, ENTCS, Edinburgh, Scotland, UK, July 2005. Elsevier.

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In E.W. Mayr and C. Puech, editors, *Proc. STACS '95*, number 900 in LNCS, pages 229–242. Springer–Verlag, 1995.

[MV94] J. McManis and P. Varaiya. Suspension automata : A decidable class of hybrid automata. In David L. Dill, editor, *CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 105–117, Stanford, CA, USA, June 1994. Springer-Verlag.

[OY96] Y. Okawa and T. Yoneda. Schedulability verification of real-time systems with extended time Petri nets. *International Journal of Mini and Microcomputers*, 18(3) :148–156, 1996.

[PH98] J.C. Palencia and M.G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 26–37, Madrid, Spain, december 1998.

[PH99] J.C. Palencia and M.G. Harbour. Exploiting precedence relations in the scheduling analysis of distributed real-time systems. In *20th IEEE Real-Time Systems Symposium (RTSS'99)*, pages 328–339, Phoenix, Arizona, USA, december 1999.

[PL00] Paul Pettersson and Kim G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70 :40–44, February 2000.

[PT99] M. Pezze and M. Toung. Time Petri nets : A primer introduction. Tutorial presented at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications, Zaragoza, Spain, september 1999.

[Ram74] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974. Project MAC Report MAC-TR-120.

[RD01] O.H. Roux and A.-M. Déplanche. Extension des réseaux de Petri t-temporels pour la modélisation de l'ordonnancement de tâches temps-réel. In *Modélisation des Systèmes Réactifs (MSR'01)*, pages 327–342, Toulouse, France, october 2001. Hermes.

[RD02] O. H. Roux and A.-M. Déplanche. A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation (JESA)*, 36(7), 2002.

[RL04] O.H. Roux and D. Lime. Time Petri nets with inhibitor hyperarcs. Formal semantics and state space computation. In *The 25th International Conference on Application and Theory of Petri Nets, (ICATPN'04)*, volume 3099 of *Lecture Notes in Computer Science*, pages 371–390, Bologna, Italy, June 2004.

[SA01] A. T. Sava and H. Alla. Commande par supervision des systèmes à évènements discrets temporisés. In *Modélisation des systèmes réactifs (MSR'01)*, pages 71–86, 2001.

[Sav01] A.T. Sava. *Sur la synthèse de la commande des systèmes à évènements discrets temporisés*. PhD thesis, Institut National polytechnique de Grenoble, Grenoble, France, november 2001.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems (extended abstract). In *13th Syposium on Theoretical Aspects of Computer Science*, pages 347–359, Grenoble, France, february 1996. Springer-Verlag.

[Tin94] K. Tindell. *Fixed priority scheduling of hard real-time systems*. PhD thesis, Department of Computer Science, University of New York, 1994.

[YR98] T. Yoneda and H. Ryuba. CTL model checking of time Petri nets using geometric regions. *IEICE Transactions on Information and Systems*, E99-D(3) :297–396, march 1998.

Annexe A

Annexes

A.1 Preuve du théorème 1

Pour tout état (M,ν) de $\mathcal{S}_{\mathcal{N}}$ et $((0,p),\bar{q},v)$ de $\mathcal{S}_{\Delta(\mathcal{N})}$ tel que $(M,\nu) \approx ((0,p),\bar{q},v)$ nous avons :

$$(M,\nu) \xrightarrow{t_i} (M',\nu') \quad ssi \quad \begin{cases} ((0,p),\bar{q},v) \stackrel{w_i}{\Longrightarrow} ((0,p'),\bar{q}',v') \ avec \\ w_i = pre_i.update.post_i.update \ et \\ (M',\nu') \approx ((0,p'),\bar{q}',v') \end{cases} \tag{A.1}$$

$$(M,\nu) \xrightarrow{d} (M',\nu') \quad ssi \quad \begin{cases} ((0,p),\bar{q},v) \xrightarrow{d} ((0,p'),\bar{q}',v') \ et \\ (M',\nu') \approx ((0,p'),\bar{q}',v') \end{cases} \tag{A.2}$$

□

Preuve. Nous prouvons d'abord la formule 5.1. Supposons $(M,\nu) \approx ((0,p),\bar{q},v)$. Alors, étant donné que t_i peut être tirée à partir de (M,ν) nous avons : (i) $M \geq {}^\bullet t_i$ (ii) $\alpha(t_i) \leq \nu_i \leq \beta(t_i)$ (iii) $M' = M - {}^\bullet t_i + t_i^\bullet$ (iv) $\nu'_k = 0$ *si* $\uparrow enabled(t_k, M, t_i)$ et $\nu'_k = \nu_k$ sinon. A partir de (i), de (ii) et de l'équivalence entre états, nous en déduisons que $\bar{q}_i = t$ et $\alpha(t_i) \leq v_i \leq \beta(t_i)$. Par conséquent, $?pre$ peut être franchie dans $\mathcal{A}_i$. Dans l'état 0 du superviseur, $!pre$ est la seule transition possible. Étant donné que la fonction de synchronisation f permet $(!pre, \bullet, \cdots, ?pre, \cdots, \bullet)$ l'action globale pre_i est possible. Après cette action $\Delta(\mathcal{N})$ atteint l'état $((1,p_1),\bar{q}_1,v_1)$ avec $\bar{q}_{1k} = \bar{q}_k, \forall k \neq i$ et $\bar{q}_{1i} = Firing$. Ainsi, $p_1 = p - {}^\bullet t_i$ et $v_1 = v$.

Maintenant, la seule transition possible pour le superviseur qui est dans l'état 1 est la transition *update* pour laquelle tous les $\mathcal{A}_i$ se synchronisent (voir f). A partir de $((1,p_1),\bar{q}_1,v_1)$ nous atteignons $((2,p_2),\bar{q}_2,v_2)$ avec $p_2 = p_1$, $v_2 = v_1$. Pour $k \neq i$, $\bar{q}_{2k} = if\, p_1 \geq {}^\bullet t_k$ et $\bar{q}_{2k} = \bar{t}$ *sinon* et $\bar{q}_{2i} = Firing$.

La transition globale suivante est nécessairement la transition $post_i$ qui conduit à $((3,p_3),\bar{q}_3,v_3)$ avec $p_3 = p_2 + t_i^\bullet$, $v_3 = v_2$ et $\bar{q}_{3k} = \bar{q}_{2k}, \forall k \neq i$ et $\bar{q}_{3i} = \bar{t}$.

A partir de ce dernier état, seule la transition *update* est possible et conduit à $((0,p_4),\bar{q}_4,v_4)$ avec $p_4 = p_3$, v_4 et q_4 donné par : $\bar{q}_{4k} = t$ si $p_3 \geq {}^\bullet t_k$ et $\bar{t}$ sinon. $v_{4k} = 0$ si $\bar{q}_{3k} = \bar{t}$ et $\bar{q}_{4k} = t$ et $v_{4k} = v_{1k}$ sinon. Notons simplement que $\bar{q}_{3k} = \bar{t}$ ssi $p - {}^\bullet t_i < {}^\bullet t_k$ et $\bar{q}_{4k} = t$ ssi $p - {}^\bullet t_i + t_i^\bullet \geq {}^\bullet t_k$. Cela implique que $v_{4k} = 0$ ssi $\uparrow enabled(t_k, p, t_i)$ et avec (iv) cela donne $\nu'_k = v_{4k}$. Étant donné que $p_4 = p_3 = p_2 + t_i^\bullet = p_1 - {}^\bullet t_i + t_i^\bullet = p - {}^\bullet t_i + t_i^\bullet$ et que (iii) nous avons $\forall i \in [1..m], M'(p_i) = p_4[i]$. Nous concluons alors que $((0,p_4),\bar{q}_4,v_4) \approx (M',\nu')$.

La réciproque de la formule 5.1 est évidente en suivant le même cheminement que précédemment.

Nous nous intéressons maintenant à la formule 5.2. D'après la sémantique des TPNs, une transition continue $(M,\nu) \xrightarrow{d} (M',\nu')$ est possible ssi $\nu = \nu' + d$ et $\forall k \in [1..n], (M \geq {}^\bullet t_k \Longrightarrow \nu'_k \leq \beta(t_k))$. D'après l'équivalence $(M,\nu) \approx ((0,p),\bar{q},v)$, si $M \geq {}^\bullet t_k$ alors $\bar{q}_k = t$ et l'évolution continue pour $\mathcal{A}_k$ est contrainte par l'invariant $x_k \leq \beta(t_k)$; dans le cas contraire, $\bar{q}_k = \bar{t}$ et l'évolution continue est non contrainte pour $\mathcal{A}_k$. Le superviseur n'étant soumis à aucune contrainte (invariant) dans l'état 0, le résultat est prouvé. □

A.2 Preuve du théorème 5

Soient $\mathcal{A}$ un TA et $\Delta(\mathcal{A})$ sa traduction en TPN telle que définie dans la section 5.4, nous avons $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\Delta(\mathcal{A}))$.

Preuve. Nous prouvons d'abord que $\Delta(\mathcal{A})$ simule (faiblement) $\mathcal{A}$ ce qui implique $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\Delta(\mathcal{A}))$. Ensuite nous montrons que nous pouvons définir un TA $\mathcal{A}'$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ et $\mathcal{A}'$ simule (faiblement) $\Delta(\mathcal{A})$ ce qui implique $\mathcal{L}(\Delta(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. Il est suffisant de prouver cela pour le cas où $\mathcal{A}$ n'a pas de transition ε. En effet dans le cas contraire, nous pouvons renommer les transitions ε en une nouvelle lettre $\mu \notin \Sigma_\varepsilon$ et obtenir ainsi un automate $\mathcal{A}_\mu$ sans transition ε. Nous appliquons alors notre traduction à $\mathcal{A}_\mu$ et obtenons un TPN dans lequel nous remplaçons les étiquettes μ par ε.

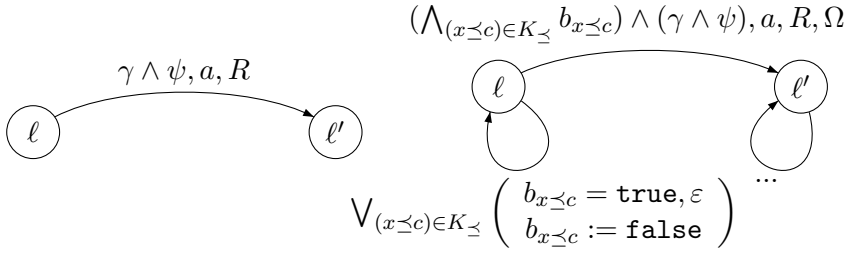
Rappelons que $\mathcal{A} = (L, \ell_0, X, \Sigma_\varepsilon, E, Inv)$ et $\Delta(\mathcal{A}) = (P, T, \Sigma_\varepsilon, {}^\bullet(.), (.)^\bullet, M_0, \Lambda, I, F_\Delta, R_\Delta)$ et notons $X = \{x_1, \cdots, x_k\}$, $P = \{p_1, \cdots, p_m\}$ et $T = \{t_1, \cdots, t_n\}$. L'ensemble des constraintes atomiques de $\mathcal{A}$ est $\mathcal{C}_\mathcal{A}$. La place γ_{tt} d'un bloc $\mathcal{N}_{x \bowtie c}$ (pour $x \bowtie c$: une contrainte atomique de $\mathcal{A}$) est notée $\gamma_{tt}^{x \bowtie c}$.

Preuve de $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\Delta(\mathcal{A}))$. Nous prouvons pour cela que $\Delta(\mathcal{A})$ simule $\mathcal{A}$. Nous définissons la relation $\sqsubseteq \subseteq (L \times \mathbb{R}^n_{\geq 0}) \times (\mathbb{N}^p \times \mathbb{R}^m_{\geq 0})$ par :

$$(\ell, v) \sqsubseteq (M, \nu) \iff \begin{cases} (1)\, M(P_\ell) = 1 \\ (2)\, \textit{pour tout } \varphi = x \bowtie c, \bowtie \in \{<, \leq\},\ M(P_u) = 0 \\ (3)\, \textit{pour tout} \varphi \in \mathcal{C}_\mathcal{A},\ v \in [\![\varphi]\!] \iff M(\gamma_{tt}^\varphi) = 1 \end{cases} \tag{A.3}$$

Prouvons que $\sqsubseteq$ est une relation de simulation faible de $\mathcal{A}$ par $\Delta(\mathcal{A})$, et ce en vérifiant les conditions de la définition Def. 7 :

1. états initiaux : il est évident que $(l_0, \overline{0}) \sqsubseteq (M_0, \overline{0})$;
2. transitions continues : soit$(\ell, v) \xrightarrow{d} (\ell, v+d)$. Considérons (M,ν) tel que $(\ell, v) \sqsubseteq (M, \nu)$. Etant donné que le bloc $\mathcal{N}_{\varphi_i}$ est non bloquant, un temps d peut s'écouler à partir de (M,ν), et il existe un run $(M,\nu) \xrightarrow{\rho} (M',\nu')$ avec $\mathit{Duration}(\mathit{trace}(\rho)) = d$ et $\mathit{Untimed}(\mathit{trace}(\rho)) = \varepsilon$. Nous pouvons choisir ρ sans transition $f(a, [0, \infty[)$; ainsi un jeton reste dans P_ℓ et $M'(P_\ell) = 1$. Pour prouver $(\ell, v+t) \sqsubseteq (M', \nu')$, il reste alors à prouver les points (2) and (3) de l'équation (A.3).
   Soit $\varphi = x \bowtie c$ avec $\bowtie \in \{<, \leq\}$.
   - si $\varphi(v) = \texttt{true}$ et $\varphi(v+d) = \texttt{true}$, il est possible de ne rien faire dans le bloc $\mathcal{N}_\varphi$ et laisser ainsi les jetons dans P_x et γ_{tt}^φ.
   - si $\varphi(v) = \texttt{true}$ et $\varphi(v+d) = \texttt{false}$, alors il existe $d' \leq d$ tel que la transition t_x du bloc $\mathcal{N}_\varphi$ est sensibilisée et doit être tirée avant que φ devienne faux. Ainsi t_x est tirée à d' puis u (immédiatement après) ce qui transfert les jetons de P_x, γ_{tt}^φ vers P_i.



(a) Arc $(\ell, \gamma \wedge \psi, a, R, \ell')$ dans $\mathcal{A}$ (b) Arc étendu dans $\mathcal{A}'$.

FIG. A.1 – De $\mathcal{A}$ vers $\mathcal{A}'$.

- si $\varphi(v) = \texttt{false}$ alors $\varphi(v+d) = \texttt{false}$, et d'après ce qui précède, il doit y avoir un jeton dans P_i et le temps peut s'écouler sans le tire d'aucune transition.

Soit $\varphi = x \bowtie c$ avec $\bowtie \in \{>, \geq\}$.

- si $\varphi(v) = \texttt{true}$ alors $\varphi(v+d) = \texttt{true}$ et $M(\gamma_{tt}^{\varphi}) = 1$. Nous avons juste à laisser le temps s'écouler dans $\mathcal{N}_\varphi$.
- si $\varphi(v) = \texttt{false}$ et $\varphi(v+d) = \texttt{true}$, alors il existe $d' \leq d$ tel que la transition t_x doit être tirée à d' (et t' peut être tirée à $d' + \xi$ avec $\xi > 0$ pour $\mathcal{N}_{x>c}$). Nous tirons t_x à d'et laissons ensuite s'écouler $d - d'$.
- si $\varphi(v) = \texttt{false}$ et $\varphi(v+d) = \texttt{false}$ alors le temps s'écoule et le jeton reste dans P_x.

De cette manière pour chaque contrainte $\varphi = x \bowtie c$, il y a un run $\rho_\varphi = (M, \nu) \xrightarrow{d}_\varepsilon (M_\varphi, \nu_\varphi)$ tel que (M_φ, ν_φ) satisfait les conditions (2) and (3) de l'équation (A.3). Prise séparément nous avons pour chaque contrainte $(\ell, v) \sqsubseteq (M_\varphi, \nu_\varphi)$. Il n'est pas difficile de construire un run ρ avec l'entrelacement des run ρ_φ précédents tel que $\rho = (M, \nu) \xrightarrow{t}_\varepsilon (M', \nu')$ et (M', ν') satisfait les conditions (2) et (3) de l'équation (A.3) pour chaque contrainte φ, et ainsi $(\ell, v) \sqsubseteq (M', \nu')$.

3. transitions discrètes : Soient $(\ell, v) \xrightarrow{a} (\ell', v')$ et $(\ell, v) \sqsubseteq (M, \nu)$. Il existe alors un arc $e = (\ell, \gamma, a, R, \ell') \in E$ tel que $\gamma = \wedge_{i=1,n} \varphi_i$, $n \geq 0$ où φ_i est une contrainte atomique. D'après la sémantique des automates temporisés (définition 17), $v \in [\![\varphi_i]\!]$ for $1 \leq i \leq n$. Cela implique $M(\gamma_{tt}^{\varphi_i}) = 1$ (par définition de la relation de simulation $\sqsubseteq$). Ainsi la transition $f(a, [0, \infty[)$ est tirée dans le bloc $\mathcal{N}_e$ conduisant à (M', ν'). Le motif $\mathcal{N}_{Reset(R)}$ remet à zéro les motifs des contraintes φ_i dont l'horloge est dans R (les marquages des autres blocs $\mathcal{N}_{\varphi_i}$ sont inchangés) et un jeton est ajouté dans $P_{\ell'}$. Le nouvel état obtenu (M'', ν'') satisfait $(\ell', v') \sqsubseteq (M'', \nu'')$.

Cela termine la preuve que $\Delta(\mathcal{A})$ simule $\mathcal{A}$ et donc que $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\Delta(\mathcal{A}))$.

Preuve de $\mathcal{L}(\Delta(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A})$. Pour cela, nous ne pouvons pas montrer une simulation évidente de $\Delta(\mathcal{A})$ par $\mathcal{A}$. En effet, dans $\Delta(\mathcal{A})$, les blocs $\mathcal{N}_{x \bowtie c}$ avec $\bowtie \in \{<, \leq\}$, imposent une décision sur le tir de la transition t_x (et u immédiatement après). Cela revient à décider qu'à partir d'un certain point $x \bowtie c$ devient faux et le restera jusqu'au prochain reset de l'horloge x. Pour établir une relation de simulation, nous allons construire un TA $\mathcal{A}'$ qui accepte le même langage que $\mathcal{A}$ mais ayant la possibilité de décider parfois (de manière non-deterministe) de ne pas utiliser une transition avec une garde $x \bowtie c$ (jusqu'au prochain reset de x). Il est alors possible de construire une relation de simulation de $\Delta(\mathcal{A})$ par $\mathcal{A}'$.

Nous notons $\preceq$ pour $\{<, \leq\}$ et $\succeq$ pour $\{>, \geq\}$. Soit $K_\preceq$ l'ensemble des contraintes $x \preceq c$ in $\mathcal{A}$. Pour chaque $x \preceq c \in K_\preceq$, nous introduisons une variable booléenne $b_{x \preceq c}$ initialement à vrai.

La construction de $\mathcal{A}'$ commence par $\mathcal{A}' = \mathcal{A}$. Nous enrichissons ensuite $\mathcal{A}'$ comme indiqué sur la figure Fig. A.1. Soit $(\ell, \gamma \wedge \psi, a, R, \ell')$ un arc de $\mathcal{A}'$ avec $\gamma = \wedge_{x \preceq c \in K_\preceq} x \preceq c$ et $\psi = \wedge_{x \succeq c \in K_\succeq} x \succeq c$. Pour ces arcs, nous renforçons[1] les gardes $\gamma \wedge \psi$ pour obtenir : $\gamma' = \gamma \wedge \psi \wedge \bigwedge_{x \preceq c \in K_\preceq} b_{x \preceq c}$. Ainsi, la transition $(\ell, \gamma \wedge \psi, a, R, \ell')$ peut être tirée dans $\mathcal{A}'$ seulement si la garde correspondante dans $\mathcal{A}$ et la conjonction des $b_{x \preceq c}$ est vraie. Nous remettons également à vrai les variables $b_{x \preceq c}$ dont $x \in R$ dans la transition $(\ell, \gamma \wedge \psi, a, R, \ell')$ et nous notons $\Omega(R) = \wedge_{x \in R} b_{x \preceq c} := \texttt{true}$.

Soit ℓ une localité de $\mathcal{A}'$. Pour chaque variable $b_{x \preceq c}$ nous ajoutons une boucle $(\ell, b_{x \preceq c} = \texttt{true}, \varepsilon, b_{x \preceq c} := \texttt{false}, \ell)$ dans $\mathcal{A}'$, ce qui signifie que l'automate $\mathcal{A}'$ peut décider de manière non-deterministe[2] de mettre $b_{x \preceq c}$ à faux (voir Fig. A.1). Il y a autant de boucle que de variables $b_{x \preceq c}$ ce qui est représenté par un $\bigvee$ dans la figure Fig. A.1. Cet automate non-deterministe $\mathcal{A}'$ accepte le même langage temporisé que $\mathcal{A}$: $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

Nous pouvons maintenant construire la relation de simulation de $\Delta(\mathcal{A})$ par $\mathcal{A}'$. Notons (ℓ, v, b) une configuration de $\mathcal{A}'$ avec b le vecteur des variables b_φ. Nous définissons la relation $\sqsubseteq \subseteq (\mathbb{N}^p \times \mathbb{R}^m_{\geq 0}) \times (L \times \mathbb{R}^n_{\geq 0} \times \mathbb{B}^k)$ par :

$$(M, \nu) \sqsubseteq (\ell, v, b) \iff \begin{cases} (1)\, M(P_\ell) = 1 \\ (2)\, \forall \varphi = x > c \in K_>,\ v \in [\![\varphi]\!] \iff M(\gamma^\varphi_{tt}) = 1 \\ (3)\, \forall \varphi = x \geq c \in K_\geq,\ v \in [\![\varphi]\!] \iff M(\gamma^\varphi_{tt}) = 1 \vee \\ \qquad\qquad (M(P^\varphi_x) = 1 \wedge \nu(t^\varphi_x) = c) \\ (4) \forall \varphi \in K_\preceq, M(P^\varphi_i) = 1 \iff (b_\varphi = \texttt{false} \vee v \notin [\![\varphi]\!]) \end{cases} \tag{A.4}$$

Maintenant, nous prouvons que $\sqsubseteq$ est une relation de simulation faible de $\Delta(\mathcal{A})$ par $\mathcal{A}$.

- concernant la configuration initiale, il est évident que $(M_0, \overline{0}) \sqsubseteq (l_0, \overline{0}, b_0)$ (dans b_0, toutes les variables b sont à vrai),
- transitions continues : soit $(M, \nu) \xrightarrow{d} (M', \nu')$ avec $d \geq 0$. Soit $(M, \nu) \sqsubseteq (\ell, v, b)$. Comme il n'y a pas d'invariant dans $\mathcal{A}'$, une durée d peut s'écouler à partir de (ℓ, v, b). Si aucune transition ε n'est tirée dans le TPN, alors toutes les valeurs de vérité des contraintes restent inchangées. Nous avons ainsi $(\ell, v, b) \xrightarrow{d} (\ell, v + d, b)$ dans $\mathcal{A}'$ avec $(M', \nu') \sqsubseteq (\ell, v + d, b)$.
- transitions discrètes : soit $(M, \nu) \xrightarrow{a} (M', \nu')$. Distinguons les 2 cas : $a = \varepsilon$ et $a \in \Sigma$.
  Si $a = \varepsilon$ alors, il s'agit soit d'une mise à jour d'un bloc $\mathcal{N}_\varphi$. (Il ne peut pas s'agir d'un reset car les reset ne sont effectués que lorsque $\sum_{\ell \in L} M(P_\ell) = 0$).
  Nous distinguons alors les cas en fonction des différents types de bloc :
  - bloc $\mathcal{N}_{x>c}$: le tir (correspondant à ce ε) est soit celui de t_x soit de t'. S'il s'agit de t_x, alors le temps qui s'est écoulé depuis le dernier reset de x est égale à c. On a alors $M(\gamma_{tt}) = 0$ et $v(x) \leq c$ et $v \notin [\![x > c]\!]$. Cela implique $(M', \nu') \sqsubseteq (\ell, v, b)$.
    En revanche s'il s'agit du tir de t', alors $v'(x) > c$ mais on a toujours $(M', \nu') \sqsubseteq (\ell, v, b)$.
  - bloc $\mathcal{N}_{x \geq c}$: on applique le même raisonnement que précédement conduisant à $(M', \nu') \sqsubseteq (\ell, v, b)$.
  - bloc $\mathcal{N}_{x<c}$: le tir (correspondant à ce ε) est soit celui de t_x soit celui de de u. S'il s'agit de t_x alors $M'(P_i) = 0$. Le temps écoulé depuis le dernier reset de x est stric-

[1] Nous avons besoin de TA étendus avec des variables booléennes ; cela n'ajoute rien au pouvoir d'expression des TA.

[2] Cela ajoute des transitions ε à $\mathcal{A}'$ mais la restriction que nous avons faite au début que $\mathcal{A}$ ne possède pas de transition ε est inutile pour montrer que $\mathcal{A}'$ simule faiblement $\Delta(\mathcal{A})$.

tement inférieur à c et $v \in [\![\varphi]\!]$. b_φ est vrai dans (ℓ, v, b) puisque $M(P_i) = 0$ et donc $(M', \nu') \sqsubseteq (\ell, v, b)$. S'il s'agit du tir de u alors $M(P_i) = 0$, $v(x) < c$ et b_φ est vrai. Nous avons $M'(P_i) = 1$. Nous choisissons dans l'automate $\mathcal{A}'$ de tirer la transition (boucle) mettant à faux la variable b_φ et obtenons un état (ℓ, v, b') tel que $(M', \nu') \sqsubseteq (\ell, v, b')$. Le même raisonnement s'applique à $\mathcal{N}_{x \leq c}$.

Si $a \in \Sigma$ alors il s'agit d'une transition $f(a, [0, \infty[)$ du bloc $\mathcal{N}_e$ de $e = (\ell, \gamma, a, R, \ell')$. Le tir de $f(a, [0, \infty[)$ ne change pas les marquages des places d'entrée γ_{tt}. D'après l'équation A.4 et la définition de $\mathcal{A}'$ nous pouvons tirer la transition correspondante $(\ell, \gamma, a, R, \ell')$ dans $\mathcal{A}'$ conduisant à l'état (ℓ', v', b'). Nous avons $M(P_\ell) = M(P'_\ell) = 0$ et cet état n'appartient pas à la relation de simulation, nous tirons alors en temps nul la séquence de transition epsilon réalisant les reset des blocs dont l'horloge $x \in R$ et conduisant à (M', ν') avec $M'(P'_\ell) = 1$. Deux cas se présentent alors :

- cette séquence de transition epsilon est composée uniquement des transitions epsilon correspondant au reset des blocs dont l'horloge $x \in R$ qui sont alors retournes dans leur état initial. Concernant les blocs $\mathcal{N}_{x \leq c}$ et $\mathcal{N}_{x < c}$, on obtient un jeton dans P_x et un dans γ_{tt} or les variables b_φ correspondantes sont repassées à vrai dans l'automate. Nous avons donc $(M', \nu') \sqsubseteq (\ell', v', b')$.
- s'ajoutant aux transition epsilon précédentes, cette séquence comprends aussi des transitions de mise à jour de bloc et le cas est alors similaire à une sequence reset suivie d'une mise à jour. Pour les mise à jours de blocs $\mathcal{N}_{x \leq c}$ et $\mathcal{N}_{x < c}$, nous tirons alors dans le TA les transitions (boucle) mettant à faux les variables b_φ correspondantes et obtenons un état (ℓ', v', b'') tel que $(M', \nu') \sqsubseteq (\ell', v', b'')$.

Cela termine la preuve que $\mathcal{A}'$ simule $\Delta(\mathcal{A})$ ainsi $\mathcal{L}(\Delta(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}')$ et donc $\mathcal{L}(\Delta(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A})$.

Nous pouvons donc conclure que $\mathcal{L}(\Delta(\mathcal{A})) = \mathcal{L}(\mathcal{A})$, ce qui termine la preuve du théorème 5. □

A.3 Preuve du théorème 6

Soient $\mathcal{A}$ un TA appartenant à la classe $\mathcal{TA}_{tpn}(\leq, \geq)$ et $\Delta(\mathcal{A})$ sa traduction en TPN telle que définie dans la section 5.5, et nous avons $\Delta(\mathcal{A}) \approx \mathcal{A}$ où $\approx$ (définition 5.5) est une relation de bisimulation faible.

Preuve. Nous prouvons que $\approx$ est une relation de bisimulation faible entre $\mathcal{A}$ et $\Delta(\mathcal{A})$.

1. concernant la configuration initiale, il est évident que $(M_0, \overline{0}) \approx (l_0, \overline{0})$,
2. transitions continues : soit$(\ell, v) \xrightarrow{d} (\ell, v+d)$. Considérons (M, ν) tel que $(\ell, v) \approx (M, \nu)$. Pour $\varphi = (x \leq c) \in Inv(\ell)$, nous avons $\varphi(v) = \texttt{true}$, $\varphi(v+d) = \texttt{true}$. Etant donné que $\nu(t_x) = v(x)$, nous avons $\nu(t_x) + d = v(x) + d \leq c$ donc $M(urg^\varphi) = 0$ et le temps d peut s'écouler dans $\mathcal{N}_\varphi$. Nous avons donc dans $\Delta(\mathcal{A})$ à partir de (M, ν) : $(M, \nu) \xrightarrow{d}_\varepsilon (M', \nu')$ avec $M(P_\ell) = M'(P_\ell) = 1$ et les évolutions des blocs suivantes :
   Pour $\varphi = (x \leq c) \in Inv(\ell)$,
   - Si $v(x) + d = \nu(t_x^\varphi) + d < c$ alors nous obtenons $M'(urg^\varphi) = 0$.
   - Si $v(x) + d = \nu(t_x^\varphi) + d = c$ alors nous obtenons soit $M(urg^\varphi) = 1$ soit $M(urg^\varphi) = 0$ et $v'(x) = \nu'(t_x^\varphi) = c$. La transition I^φ est soit sensibilisée soit sera sensibilisée apres le tir immédiat de t_x, interdisant ainsi l'écoulement du temps tant que $M(P_\ell) = 1$.

   Pour $\varphi = x \geq c$,
   - $\varphi(v) = \texttt{true}$ et $\varphi(v+d) = \texttt{true}$. Si $M(\gamma_{tt}^\varphi) = 1$ le temps s'écoule dans $\mathcal{N}_\varphi$. Si $M(\gamma_{tt}^\varphi) = 0$ alors $M(P_x^\varphi) = 1$ et (puisque $d > 0$) t_x^φ est tirée avant l'écoulement de d.

- $\varphi(v) = \texttt{false}$ et $\varphi(v+d) = \texttt{true}$, ssi il existe $d' \leq d$ tel que la transition t_x doit être tirée à d'. La transition t_x est donc tirée à d' puis il s'écoule $d - d'$.
- $\varphi(v) = \texttt{false}$ et $\varphi(v+d) = \texttt{false}$ ssi le temps s'écoule dans $\mathcal{N}_\varphi$ et le jeton reste dans P_x.

Pour $\varphi = (x \leq c) \notin Inv(\ell)$, d'après la sous classe des TA considérée, il s'agit d'une contrainte qui ne sera plus utilisée avant le prochain reset de x .

- $\varphi(v+d) = \texttt{false}$. Si $M(urg^\varphi) = 1$ alors le temps d peut s'écouler. Si $M(urg^\varphi) = 0$ alors il existe $d' \leq d$ tel que la transition t_x doit être tirée à d'. t_x est donc tirée à d' puis il s'écoule $d - d'$.
- $\varphi(v) = \texttt{true}$ et $\varphi(v+d) = \texttt{true}$. Alors le cas est identique à si $\varphi \in Inv(\ell)$ mais la transition I^φ ne sera pas sensibilisé car (P_ℓ) n'est pas une de ses places d'entrée.

De cette manière, pour chaque contrainte, tous les run $\rho_\varphi = (M, \nu) \xrightarrow{d}_\varepsilon (M_\varphi, \nu_\varphi)$ sont tels que (M_φ, ν_φ) satisfait les conditions (2), (3) et (4) de l'équation (5.5). Quelquesoit l'entrelacement des run ρ_φ précédents nous obtenons $\rho = (M, \nu) \xrightarrow{d}_\varepsilon (M', \nu')$ tel que $(\ell, v) \approx (M', \nu')$.

3. transitions discrètes : soient $(\ell, v) \xrightarrow{a} (\ell', v')$ et $(\ell, v) \approx (M, \nu)$. Il existe alors un arc $e = (\ell, \gamma, a, R, \ell') \in E$ tel que $\gamma = \wedge_{i=1,n}\varphi_i$, $n \geq 0$ où φ_i est une contrainte atomique. Etant donnée la sous classe des TA considérée, les invariants de ℓ' ne sont pas à prendre en compte pour autoriser le tir de a car ils sont vrais par définition si les invariants de ℓ sont vrais. D'après la sémantique des automates temporisés (définition 17), $v \in [\![\varphi_i]\!]$ for $1 \leq i \leq n$. Par définition de la relation de bisimulation $\approx$ nous avons donc, soit $M(\gamma_{tt}^{\varphi_i}) = 1$, soit $M(\gamma_{tt}^{\varphi_i}) = 0$ et la transition $t_x^{\varphi_i}$ est tirable immédiatement conduisant à $M(\gamma_{tt}^{\varphi_i}) = 1$. Ainsi la transition $f(a, [0, \infty[)$ est tirée dans le bloc $\mathcal{N}_e$ conduisant à (M', ν'). On a alors $M'(P_\ell) = M'(P'_\ell) = 0$ et $\Delta(\mathcal{A})$ ne peut rien faire d'autre que des transitions epsilon en temps nul $(M', \nu') \xrightarrow{0}_\varepsilon (M'', \nu'')$ jusqu'à $M''(P_\ell) = 1$. Le motif $\mathcal{N}_{Reset(R)}$ remet à zéro les motifs des contraintes φ dont l'horloge x est dans R ce qui donne $M''(P_x^\varphi) = 1$ et donc $\nu''(t_x^\varphi) = v'(x) = 0$. Pendant cette phase de reset, si une transition t_x^φ est tirée,
   - soit $x \in R$ auquel cas t_x^φ a été tiré avant le reset du bloc $\mathcal{N}_\varphi$. A l'issu de la phase de reset, nous avons donc $M''(P_x^\varphi) = 1$,
   - soit $x \notin R$ auquel cas $\nu''(t_x^\varphi) = v'(x) = c$. Nous obtenons donc dans le bloc $\mathcal{N}_\varphi$ $M''(\gamma_{tt}^\varphi) = 1$ ce qui satisfait l'équation 3,

   Le nouvel état obtenu (M'', ν'') satisfait $(\ell', v') \approx (M'', \nu'')$. A partir de (M'', ν''), tout tir d'une transition espilon $(M'', \nu'') \xrightarrow{0}_\varepsilon (M''', \nu'')$ est une transition t_x^φ qui entre dans le 2^eme cas "$x \notin R$" précédent donc $(\ell', v') \approx (M'', \nu'')$

Cela termine la preuve que $\approx$ est une relation de bisimulation. □

A.4 Preuve du théorème 13

Soient $\mathcal{N}$, un *Scheduling*-TPN et $\mathcal{A} = (L, l_0, X, A, E, Inv, Dif)$, son automate à chronomètre des classes d'états défini dans la section 7.3.2. Soient $Q_{\mathcal{N}}$ l'ensemble des états du réseau $\mathcal{N}$ et $Q_{\mathcal{A}}$ l'ensemble des états de $\mathcal{A}$. Soit $\mathcal{R} \subset Q_{\mathcal{N}} \times Q_{\mathcal{A}}$ une relation binaire telle que $\forall s = (M_{\mathcal{N}}, \nu_{\mathcal{N}}) \in Q_{\mathcal{N}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}$,

$$s\mathcal{R}a \Leftrightarrow \begin{cases} M_{\mathcal{N}} = {M_{\mathcal{A}}}^3 \\ \forall t \in enabled(M_{\mathcal{N}}), \exists x \in X \text{ t.q. } \nu_{\mathcal{N}}(t) = \nu_{\mathcal{A}}(x) \text{ et } \begin{cases} \dot{x} = 1 \text{ si } t \text{ est active} \\ \dot{x} = 0 \text{ sinon} \end{cases} \\ \forall x \in X, \exists t \in enabled(M_{\mathcal{N}}) \text{ t.q. } \nu_{\mathcal{N}}(t) = \nu_{\mathcal{A}}(x) \text{ et } t \text{ est active ssi } \dot{x} = 1 \end{cases}$$

$\mathcal{R}$ est une bisimulation.

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

1. Supposons que le *Scheduling*-TPN puisse laisser s'écouler le temps $d \in \mathbb{R}^+ : s \xrightarrow{d} s'$. Cela signifie que $\forall t \in enabled(M_{\mathcal{N}})$, $\nu_{\mathcal{N}}(t) + d \leq \beta(t)$. Donc, $\nu_{\mathcal{A}}(x_t) + d \leq \beta(t)$ et donc par définition de $Inv(l)$, $Inv(l)(\nu_{\mathcal{A}} + d')$ est vrai $\forall d' \leq d$. Par conséquent, le SWA $\mathcal{A}$ peut laisser le temps d s'écouler : $a \xrightarrow{d} a'$. Puisque le *Scheduling*-TPN reste dans le même marquage, et le SWA dans la même localité, l'activité des transitions et les conditions sur $\dot{x}$ ne changent pas. Par conséquent, $\nu_{\mathcal{N}} + d = \nu_{\mathcal{A}} + d$ et finalement $s'\mathcal{R}a'$.

2. Supposons que le *Scheduling*-TPN puisse tirer la transition $t \in T : s \xrightarrow{t} s'$. Par définition de l'automate à chronomètres des classes d'états, il existe une arête $e = (l, \delta, t, R, \rho, l')$. Cela signifie que $\alpha(t) \leq \nu_{\mathcal{N}}(t)$. Donc, $\alpha(t) \leq \nu_{\mathcal{A}}(x_t)$ et par définition de la garde δ, $\delta(\nu_{\mathcal{A}})$ est vérifiée. Donc le SWA $\mathcal{A}$ peut prendre l'arête $e : a \xrightarrow{t} a'$. Par définition de l', le marquage $M'_{\mathcal{A}}$ de la classe d'états étendue l' est le même que le nouveau marquage $M'_{\mathcal{N}}$ du *Scheduling*-TPN. Soit t' une transition de $enabled(M'_{\mathcal{N}})$.
   Si t' est nouvellement sensibilisée, un nouveau chronomètre est créé où t' est associée à un chronomètre dont la valeur est 0 et dont l'activité est la même que celle de t'. Dans le premier cas, l'activité du chronomètre est également définie en accord avec l'activité de t'. Si t' n'est pas nouvellement sensibilisée, et si toutes les transitions associées à son chronomètre ont la même activité, alors l'activité du chronomètre est redéfinie en accord avec l'activité de t', sinon, si toutes les transitions associées à ce chronomètre n'ont pas la même activité, un nouveau chronomètre est créé dont l'activité est celle de t' et auquel t' est associée. Finalement, $s'\mathcal{R}a'$.

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

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

□