Université de Nantes

École Doctorale

Sciences et Technologies de l'Information et Mathématiques

Année : 2004

Habilitation à diriger des recherches de l'Université de Nantes

Spécialité : Automatique et Informatique Appliquée

Présentée et soutenue publiquement par :

Didier Lime

le 12 décembre 2012
à l'École Centrale de Nantes

Titre :

Contributions à l'Analyse des Systèmes Temporisés, Ouverts et Répartis

Jury

Rapporteurs	Hanifa Boucheneb	Professeure à l'École Polytechnique de Montréal, Canada
	Patricia Bouyer-Decitre	Directrice de recherche CNRS - LSV, Cachan
	François Laroussinie	Professeur à l'Université Paris-Diderot, LIAFA
Examinateurs	Claude Jard	Professeur à l'Université de Nantes, LINA
	Jan Komenda	Chercheur à l'Académie des Sciences, Brno, Rép. Tchèque
	Olivier H. Roux	Professeur à l'École Centrale de Nantes, IRCCyN
	François Vernadat	Professeur à l'INSA Toulouse, LAAS

Directeur de recherches : Olivier H. Roux

Table des matières

Avant-propos

Ce mémoire présente mes travaux de recherche qui portent principalement sur l'étude des modèles formels pour les systèmes temporisés. Ce thème fédérateur se décline selon plusieurs axes : analyse d'expressivité, extensions des modèles classiques, conception d'algorithmes et de méthodes de vérification automatique et mise en œuvre de ces techniques au travers de l'implémentation dans des outils logiciels.

Ce document est composé de deux parties principales : la première présente une synthèse de mes activités et propose une vue générale de mes travaux de recherche, et la seconde développe de façon beaucoup plus détaillée et technique deux thèmes choisis parmi ces contributions.

Première partie

Synthèse des activités

Chapitre 1

Curriculum Vitæ

Je présente dans ce chapitre quelques éléments factuels reflétant mon activité, principalement de recherche. Le contexte scientifique et l'articulation de ces éléments seront présentés plus avant dans le chapitre 2.

1.1 Situations professionnelles depuis la thèse

Depuis 2005	**Maître de conférences à l'École Centrale de Nantes**. Enseignement : département Informatique et Mathématiques ; recherche : IRCCyN.
2004-2005	**Post-doctorant à l'Université d'Aalborg, Danemark**. Avec Kim G. Larsen.

1.2 Diplômes universitaires

2004	**Doctorat en automatique et informatique appliquée** de l'Université de Nantes : *Vérification d'applications temps réel à l'aide de réseaux de Petri temporels étendus*, encadré par Olivier H. Roux (95%) et Yvon Trinquet (5%) et soutenu le 1er décembre 2004. Jury : J. Sifakis, S. Haddad, G. Juanole, B. Berthomieu, O.H. Roux, Y. Trinquet, J.-L. Ferrier.
2001	**DEA en automatique et informatique appliquée** de l'École Centrale de Nantes.
2001	**Diplôme d'ingénieur généraliste** de l'École Centrale de Nantes, option *informatique*.

1.3 Expérience pédagogique

2007	**École Nationale de la Statistique et de l'Analyse de l'Information (ENSAI)**. Enseignements des *systèmes d'exploitation*.
Depuis 2006	**Master recherche automatique et systèmes de production (ASP)**. Enseignements de *modélisation et vérification*.
Depuis 2005	**École centrale de Nantes**. Enseignements d'*algorithmique et programmation*, *méthodes logicielles*, *C++*, *systèmes d'exploitation*, *compilation* et *décidabilité et complexité (introduction)*.
2001-2004	**École centrale de Nantes**. Enseignements d'*algorithmique et programmation* et *systèmes temps réels*.
2001-2004	**École des mines de Nantes**. Enseignements *programmation fonctionnelle* et *systèmes et réseaux*.

1.4 Publications et production scientifique

1.4.1 Publications

L'ordre des noms est alphabétique sauf lorsque la publication concerne le travail de doctorants. Dans ce cas, leur nom se trouve en tête.

Revues internationales

[J1] Gilles Benattar, Béatrice Bérard, Didier Lime, John Mullins, Olivier H. Roux, and Mathieu Sassolas. Channel Synthesis for Finite Transducers *International Journal of Foundations of Computer Science*, 23(6):12–41, 2012. World Scientific Publishing Company.

[J2] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. Parametric model-checking of stopwatch Petri nets. *Journal of Universal Computer Science (J.UCS)*, 15(17):3273–3304, 2009. Graz University of Technology et Universiti Malaysia Sarawak.

[J3] Didier Lime and Olivier H. Roux. Formal verification of real-time systems with preemptive scheduling. *Journal of Real-Time Systems*, 41(2):118–151, 2009. Springer.

[J4] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. When are timed automata weakly timed bisimilar to time Petri nets ? *Theoretical Computer Science*, 403(2-3):202–220, 2008. Elsevier.

[J5] Bernard Berthomieu, Didier Lime, Olivier H. Roux, and François Vernadat. Reachability problems and abstract state spaces for time Petri nets with stopwatches. *Journal of Discrete Event Dynamic Systems (jDEDS)*, 17(2):133–158, 2007. Kluwer.

[J6] Jamil Ahmad, Gilles Bernot, Jean-Paul Comet, Didier Lime, and Olivier Roux. Hybrid modelling and dynamical analysis of gene regulatory networks with delays. *ComPlexUs*, 3(4):231–251, 2006.

[J7] Didier Lime and Olivier H. Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Event Dynamic Systems (jDEDS)*, 16(2):179–205, April 2006. Kluwer.

Revues francophones

[F1] Didier Lime, Claude Martinez, and Olivier H. Roux. Coercition temporelle de réseaux de Petri. *Journal Européen des Systèmes Automatisés (JESA) - Numéro spécial « Actes du 8^e Colloque Francophone sur la Modélisation des Systèmes Réactifs (MSR'11) »*, 45(1-2-3):13–28, November 2011.

[F2] Didier Lime and Olivier H. Roux. Vérification formelle des systèmes temps réel avec ordonnancement préemptif. *Techniques et Sciences Informatiques (TSI)*, 25(3):347–375, 2006.

[F3] 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 (JESA) - Numéro spécial « Actes du 5^e Colloque Francophone sur la Modélisation des Systèmes Réactifs (MSR'05)"*, 39(1-2-3):223–238, October 2005.

Conférences internationales avec comité de lecture et actes publiés

[C1] Aleksandra Jovanović, Sébastien Faucou, Didier Lime and Olivier H. Roux. Real-Time Control with Parametric Timed Reachability Games. In *11th International Workshop on Discrete Event Systems (WODES'12)*, pages 323–330, Guadalajara, Mexico, October 2012. IFAC.

[C2] S Akshay, Loïc Hélouët, Claude Jard, Didier Lime and Olivier H. Roux. Robustness of Time Petri Nets under Architectural Constraints. In Marcin Jurdziński and Dejan Nickovic editors, *10th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2012)*, volume 7595 of *Lecture Notes in Computer Science*, pages 11–26, London, UK, September 2012. Springer.

[C3] Timothy Bourke, Alexandre David, Kim. G. Larsen, Axel Legay, Didier Lime, Ulrik Nyman, and Andrzej Wasowski. New results on time specifications. In Till Mossakowski and Hans-Jörg Kreowski, editors, *Recent Trends in Algebraic Development Techniques. Revised selected papers from the 20th International Workshop on Algebraic Development Techniques in Verification, Model Checking, and Abstraction Interpretation (WADT'10)*, volume 7137 of *Lecture Notes in Computer Science*, pages 175–192, February 2012. Springer.

[C4] Gilles Benattar, Béatrice Bérard, Didier Lime, John Mullins, Olivier H. Roux, and Mathieu Sassolas. Channel synthesis for finite transducers. In *13th International Conference on Automata and Formal Languages (AFL 2011)*, pages 79–92, Debrecen, Hungary, August 2011. Institute of Mathematics and Computer Science of Nyíregyháza College.

[C5] Yann Thierry-Mieg, Béatrice Bérard, Fabrice Kordon, Didier Lime, and Olivier H. Roux. Compositional analysis of discrete time Petri nets. In *1st workshop on Petri Nets Compositions (CompoNet 2011)*, volume 726, pages 17–31, Newcastle, UK, June 2011. CEUR.

[C6] Louis-Marie Traonouez, Bartosz Grabiec, Claude Jard, Didier Lime, and Olivier H. Roux. Symbolic unfolding of parametric stopwatch Petri nets. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *8th International Symposium on Automated Technology*

for Verification and Analysis (ATVA 2010), volume 6252 of *Lecture Notes in Computer Science*, pages 291–305, Singapore, September 2010. Springer.

[C7] Bartosz Grabiec, Louis-Marie Traonouez, Claude Jard, Didier Lime, and Olivier H. Roux. Diagnosis using unfoldings of parametric time Petri nets. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *8th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2010)*, volume 6246 of *Lecture Notes in Computer Science*, pages 137–151, Vienna, Austria, September 2010. Springer.

[C8] Gilles Benattar, Franck Cassez, Didier Lime, and Olivier H. Roux. Synthesis of non-interferent timed systems. In Joël Ouaknine and Frits Vaandrager, editors, *7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2009)*, volume 5813 of *Lecture Notes in Computer Science*, pages 28–42, Budapest, Hungary, September 2009. Springer.

[C9] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou, editors, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57, York, United Kingdom, March 2009. Springer.

[C10] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. Parametric model-checking of time Petri nets with stopwatches using the state-class graph. In Franck Cassez and Claude Jard, editors, *6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2008)*, volume 5215 of *Lecture Notes in Computer Science*, pages 280–294, Saint-Malo, France, September 2008. Springer.

[C11] Morgan Magnin, Didier Lime, and Olivier H. Roux. Symbolic state space of stopwatch Petri nets with discrete-time semantics. In Kees van Hee and Rüdiger Valk, editors, *The 29th International Conference on Application and Theory of Petri Nets and other models of concurrency (ICATPN 2008)*, volume 5062 of *Lecture Notes in Computer Science*, pages 307–326, Xi'an, China, June 2008. Springer.

[C12] Franck Cassez, Alexandre David, Kim G. Larsen, Didier Lime, and Jean-François Raskin. Timed control with observation based and stuttering invariant strategies. In *5th International Symposium on Automated Technology for Verification and Analysis (ATVA 2007)*, volume 4762 of *Lecture Notes in Computer Science*, pages 192–206, Tokyo, Japan, October 2007. Springer.

[C13] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Uppaal-tiga: Time for playing games ! In *19th International Conference on Computer Aided Verification (CAV 2007)*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125, Berlin, Germany, July 2007. Springer.

[C14] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. When are timed automata weakly timed bisimilar to time Petri nets ? In R. Ramanujam and Sandeep Sen, editors, *25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2005)*, volume 3821 of *Lecture Notes in Computer Science*, pages 273–284, Hyderabad, India, December 2005. Springer-Verlag.

[C15] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of different semantics for time Petri nets. In Doron A. Peled and Yih-Kuen Tsay, editors, *3rd International Symposium on Automated Technology for Verification*

and Analysis (ATVA 2005), volume 3707 of *Lecture Notes in Computer Science*, pages 293–307, Taipei, Taiwan, October 2005. Springer-Verlag.

[C16] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of the expressiveness of timed automata and time Petri nets. In Paul Pettersson and Wang Yi, editors, *3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2005)*, volume 3829 of *Lecture Notes in Computer Science*, pages 211–225, Uppsala, Sweden, September 2005. Springer-Verlag.

[C17] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *16th International Conference on Concurrency Theory (CONCUR 2005)*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80, San Fransisco, CA, USA, August 2005. Springer-Verlag.

[C18] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Roméo: A tool for analyzing time Petri nets. In Kousha Etessami and Sriram K. Rajamani, editors, *17th International Conference on Computer Aided Verification (CAV 2005)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK, July 2005. Springer-Verlag.

[C19] Morgan Magnin, Didier Lime, and Olivier H. Roux. An efficient method for computing the exact state space of Petri nets with stopwatches. In *3rd International Workshop on Software Model-Checking (SoftMC 2005)*, volume 144(3) of *Electronic Notes in Theoretical Computer Science*, pages 59–77, Edinburgh, Scotland, UK, July 2005. Elsevier.

[C20] Didier Lime and Olivier 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)*, pages 187–196, Lisbon, Portugal, December 2004. IEEE Computer Society Press.

[C21] Olivier H. Roux and Didier Lime. Time Petri nets with inhibitor hyperarcs. Formal semantics and state space computation. In Jordi Cortadella and Wolfgang Reisig, editors, *The 25th International Conference on Application and Theory of Petri Nets (ICATPN 2004)*, volume 3099 of *Lecture Notes in Computer Science*, pages 371–390, Bologna, Italy, June 2004. Springer-Verlag.

[C22] Didier Lime and Olivier H. Roux. State class timed automaton of a time Petri net. In *10th International Workshop on Petri Nets and Performance Models, (PNPM 2003)*, pages 124–133, Urbana, USA, September 2003. IEEE Computer Society.

[C23] 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 2003)*, Aveiro, Portugal, July 2003. Elsevier Science.

Ouvrages en tant qu'éditeur

[E1] Olivier H. Roux and Didier Lime, editors. *European Journal of Automation (JESA) - Special Issue in french on Modélisation des Systèmes Réactifs*, volume 43(7-8-9). Hermes Lavoisier, November 2009.

[E2] Sébastien Faucou, Didier Lime, and Olivier H. Roux, editors. *Proceedings of the Summer School ETR'2007*. IRCCyN, Nantes, France, September 2007.

Chapitres dans des ouvrages en anglais

[B1] Didier Lime, Olivier H. Roux, and Jiří Srba. Models for real-time embedded systems. In Olivier H. Roux and Claude Jard, editors, *Communicating Embedded Systems – Software and Design*, pages 1–37. ISTE Publishing / John Wiley, October 2009.

[B2] Alexandre David, Gerd Behrmann, Peter Bulychev, Joakim Byg, Thomas Chatain, Kim G. Larsen, Paul Pettersson, Jacob I. Rasmussen, Jiří Srba, Wang Yi, Kenneth Y. Joergensen, Didier Lime, Morgan Magnin, Olivier H. Roux, and Louis-Marie Traonouez. Tools for model-checking timed systems. In Olivier H. Roux and Claude Jard, editors, *Communicating Embedded Systems – Software and Design*, pages 165–225. ISTE Publishing / John Wiley, October 2009.

[B3] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of expressiveness for timed automata and time Petri nets. In Vangelis Th. Paschos, editor, *Combinatorial Optimization and Theoretical Computer Science*, pages 93–144. ISTE Publishing / John Wiley, January 2008.

Chapitres dans des ouvrages en français

[L1] Didier Lime and Olivier H. Roux. Les modèles pour les systèmes temps réel embarqués. In Olivier H. Roux and Claude Jard, editors, *Approches formelles des systèmes embarqués communicants*, pages 45–74. Traité IC2, Hermes Lavoisier, 2008.

[L2] Alexandre David, Gerd Behrmann, Kim G. Larsen, Paul Pettersson, Jacob I. Rasmussen, Wang Yi, Didier Lime, Morgan Magnin, and Olivier H. Roux. Outils de model-checking. In Olivier H. Roux and Claude Jard, editors, *Approches formelles des systèmes embarqués communicants*, pages 199–244. Traité IC2, Hermes Lavoisier, 2008.

Workshops internationaux avec comité de lecture et actes informels

[W1] Didier Lime, Olivier H. Roux and Claude Jard. Clock Transition Systems. In *21th international Workshop on Concurrency, Specification and Programming (CS&P 2012)*. Berlin, Germany, September 2012.

[W2] Gilles Benattar, Béatrice Bérard, Didier Lime, John Mullins, Olivier H. Roux, and Mathieu Sassolas. Covert channels with sequential transducers. In *8th Workshop on Foundations of Computer Security (FCS 2009)*, August 2009.

[W3] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. UPPAAL-Tiga: Timed Games for Everyone. In Luca Aceto and Anna Ingolfdottir, editors, *Proceedings of the 18th Nordic Workshop on Programming Theory (NWPT'06), Reykjavik, Iceland*. Reykjavik University, 2006.

Mémoires

[M1] Didier Lime. *Vérification d'applications temps réel à l'aide de réseaux de Petri temporels étendus*. PhD thesis, University of Nantes, Nantes, France, December 2004.

1.4.2 Développement logiciel

Je participe au développement de deux outils logiciels : Roméo et Uppaal-TiGa.

Roméo

Roméo est un outil permettant la saisie, la simulation, le calcul de l'espace d'états, la vérification et le contrôle des réseaux de Petri temporels et de leur extension à chronomètres ou paramétrées.

J'ai écrit le module de calcul actuel en 2001, depuis étendu notamment par les contributions de quatre doctorants. Ce module représente environ 35 000 lignes de C++.

Roméo reste un prototype mais a été au centre de plusieurs relations avec des industriels formalisées par des contrats (Dassault, Sodius) ou non (EADS). Le logiciel est disponible gratuitement, sous license CECILL, pour Windows, Linux et Mac OS X à l'adresse `http://romeo.rts-software.org`.

Uppaal-Tiga

Uppaal-TiGa est une version de l'outil Uppaal dédiée au problème du contrôle temporisé. Il a été le premier outil efficace de synthèse de contrôleurs temporisés et reste le plus abouti.

J'ai écrit en C++ la première version du logiciel lors de mon post-doctorat à Aalborg. Depuis, l'outil a été grandement amélioré et continue à progresser, en particulier grâce à des stagiaires de l'École Centrale de Nantes que j'envoie à Aalborg depuis 2006. Le logiciel est disponible gratuitement pour Windows, Linux et Mac OS X à l'adresse `http://www.cs.aau.dk/~adavid/tiga/index.html`. De 2006 à 2011, j'ai maintenu la version MacOS X.

1.5 Encadrement

1.5.1 DEA et Master recherche

- Christelle Hobeika. *Vérification et contrôle des automates temporisés à coûts*. 2006.
- Morgan Magnin. *Vérification de réseaux de Petri temporels étendus à l'aide de polyèdres*. 2004.

1.5.2 Doctorats

- **Gilles Benattar**, thèse soutenue le 6 avril 2011
  Synthèse de systèmes informatiques non interférents
  Encadrement : Didier Lime (80%) et Olivier H. Roux (20%)
  Financement : Bourse MESR
  Jury : S. Haddad, T. Jéron, L. Helouët, C. Kirchner, D. Lime, O.H. Roux
  Publications associées : 1 revue internationale [J1], 2 conférences internationales [C8, C4] et un workshop international [W2].
  Situation actuelle : Ingénieur R&D « méthodes formelles », ClearSy, Paris.

- **Aleksandra Jovanović**, thèse commencée le 1er octobre 2010
  Implémentation robuste des systèmes temporisés
  Encadrement [1] : Sébastien Faucou (30%), Didier Lime (30%) et Olivier H. Roux (40%)
  Financement : Bourse MESR

1. Encadrement effectif sur 2010-2012 : D. Lime (80%), O.H. Roux (20%)

Publications associées : [C1] plus un article soumis à une conférence internationale

De plus, j'ai été très fortement impliqué dans l'encadrement de Louis-Marie Traonouez à partir de sa deuxième année de thèse, en accompagnement d'une focalisation plus importante sur les dépliages et la vérification paramétrés :

– **Louis-Marie Traonouez**, thèse soutenue le 27 novembre 2010
Vérification et dépliages de réseaux de Petri temporels paramétrés
Encadrement [2] : David Delfieu et Olivier H. Roux
Financement : Bourse MESR
Jury : C. Jard, F. Laroussinie, J.-L. Boimond, B. Berthomieu, D. Delfieu, O.H. Roux
Publications associées à mon encadrement : 1 revue internationale [J2] et 4 conférences internationales [C10, C9, C7, C6]
Situation actuelle : Postdoctorant à Aalborg (avec Kim G. Larsen), Danemark.

1.6 Rayonnement Scientifique

1.6.1 Collaborations nationales

– avec Bernard Berthomieu et François Vernadat du LAAS (Toulouse), sur l'étude de problèmes de décidabilité sur les les réseaux de Petri temporels étendus avec des chronomètres (« stopwatches »). Ce qui a conduit aux publications [J5, F3] ;
– avec Béatrice Bérard (LIP6, UPMC, Paris) et Serge Haddad (LSV, ENS de Cachan) sur la comparaison de l'expressivité des automates temporisés et des réseaux de Petri temporels. Notre collaboration a conduit aux publications [C16, C14, C15, B3] ;
– avec Béatrice Bérard, Fabrice Kordon et Yann Thierry-Mieg du LIP6 sur une approche symbolique à base de diagrammes de décisions pour la vérification de systèmes en temps discrets, publiée dans [C5].
– avec Béatrice Bérard et Mathieu Sassolas du LIP6 sur la modélisation et la détection de canaux cachés à l'aide de transducteurs rationels, en lien avec la thèse de Gilles Benattar. Cela a conduit aux publications [W2, C4, J1].
– avec Claude Jard et Bartosz Grabiec de l'IRISA à Rennes sur le dépliage de réseaux de Petri temporels et la supervision, en lien avec la thèse de Louis-Marie Traonouez. Ces travaux ont été publiés dans [C7, C6].
– plus ponctuellement, j'ai travaillé avec Gilles Bernot et Jean-Paul Comet de l'I3S (Nice) [J6] ainsi qu'avec Axel Legay et Timothy Bourke (IRISA, Rennes) [C3].

1.6.2 Collaborations internationales

– avec Kim G. Larsen et Alexandre David (Université d'Aalborg, Danemark). Nos travaux portent sur la synthèse de contrôleurs pour des systèmes temporisés. J'ai passé un an à Aalborg, lors de mon post-doctorat. J'y ai été invité 15 jours en mai 2006, ainsi qu'une semaine en septembre 2008 et en septembre 2009. J'ai également invité Alexandre David à l'IRCCyN, 15 jours en janvier 2007. Depuis 2006, j'ai envoyé huit étudiants de l'École Centrale de Nantes à Aalborg pour effectuer leur stage de fin d'études encadré par Alexandre David. Cette collaboration a en outre aux publications suivantes [C17, C12, W3, C13, L2, B2, C3].

2. Encadrement effectif sur 2008-2010 : D. Lime (50%), O.H. Roux (50%)

- avec John Mullins (École Polytechnique de Montréal, Canada). Ces travaux portent sur la synthèse de contrôleurs temporisés assurant des propriétés de non-interférence et sur la modélisation et la détection de canaux cachés à l'aide de transducteurs rationels (en lien avec la thèse de Gilles Benattar). J'ai été invité à Montréal 15 jours en mai 2007 et 10 jours en février 2009 et j'ai invité John Mullins sur un poste de professeur invité à l'École centrale de Nantes en avril 2008, dans mon département d'enseignement. Cela a déjà conduit à trois publications : [W2, C4, J1]. En 2011 et 2012, j'ai envoyé un étudiant de l'École Centrale de Nantes à Montréal pour effectuer son stage de fin d'études encadré par John Mullins.
- Plus ponctuellement, j'ai travaillé avec Jean-François Raskin (Université Libre de Bruxelles, Belgique) [C12], Jiri Srba (Université d'Aalborg, Danemark) [B1] et Andrzej Wasowski (ITU Copenhague, Danemark) [C3].

1.6.3 Comités de programme et évaluation d'articles

- Je suis ou j'ai été dans les comités de programmes des manifestations suivantes : TiSto'09, DOTS'10 (co-chair), ETR'11 et ETR'13.
- Je suis régulièrement sollicité pour évaluer des articles pour des conférences (dont ICATPN, TACAS, ICALP, CONCUR, FORMATS, QEST, LATA, HSCC, ACSD et SOFSEM) et pour les revues suivantes :
  - Theoretical Computer Science ;
  - Journal of Real-time Systems ;
  - Journal of Discrete Event Dynamic Systems ;
  - International Journal of Computer Mathematics ;
  - International Journal of Foundations of Computer Science ;
  - IEEE Transactions on Software Engineering ;
  - IEEE Transactions on Control Systems Technology ;
  - IEEE Transactions on Industrial Informatics ;
  - Journal of Systems and Software ;
  - Techniques et Sciences Informatiques ;
  - Information and Computation.

1.6.4 Expertises

- depuis 2010, je suis expert pour la partie 3 (aspects compositionnels et extensions) de la *norme ISO/IEC-15909* sur les réseaux de Petri ;
- en 2009, j'ai été évaluateur pour l'appel à projet *logiciel et systèmes complexes* de la région Île-de-France et de DIGITEO.

1.6.5 Jurys de thèse

- examinateur de la thèse de Bartosz Grabiec, *Supervision of distributed systems using constrained unfoldings of timed models*, octobre 2011, encadré par Claude Jard (IRISA, Rennes) ;
- examinateur de la thèse d'Omer Landry Nguena Timo, *Synthèse pour une logique temps réel faible*, décembre 2009, encadré par Igor Walukiewicz (LaBRI, Bordeaux).

1.6.6 Comité de sélection

Membre extérieur du comité de sélection du poste 0840 à l'Université de Bordeaux 1 (2011).

1.6.7 Autres

- Titulaire de la Prime d'Encadrement Doctoral et de Recherche (PEDR) depuis 2008.
- Qualification obtenue en 27e et 61e sections avec les numéros de qualification suivants :

Section	Numéro de qualification	Date de qualification
27	05227152884	28/01/2005
61	05261152884	02/02/2005

1.7 Responsabilités scientifiques

1.7.1 Responsabilités d'enseignement

- Responsable pédagogique du cours d'*algorithmique et programmation* (tronc commun 1ère année tronc commun, 2006–2010) ;
- Responsable pédagogique du cours d'*algorithmique et structures de données* (filière par apprentissage, depuis 2010) ;
- Responsable pédagogique du cours de *systèmes d'exploitation* (3e année option informatique, depuis 2006) ;
- Responsable pédagogique du cours de *compilation* (3e année option informatique, depuis 2009) ;
- Responsable des stages de fin d'études des étudiants de 3e année option informatique (2007–2011) ;
- Responsable des projets d'application de l'option informatique (depuis 2011) ;

1.7.2 Responsabilités de recherche

- Chargé de mission *valorisation* pour l'IRCCyN (depuis janvier 2012).
- Membre élu au conseil scientifique de l'École centrale de Nantes (depuis 2010).
- Membre élu au conseil de l'IRCCyN (2008–2011) ;

1.7.3 Organisation de manifestations scientifiques

- En 2011, j'ai coorganisé avec François Vernadat la session *méthode formelles*, de la 7e *école d'été temps réel* (ETR'11) organisée à Brest ;
- En 2010, j'ai coorganisé avec Béatrice Bérard, à Paris en satellite de CONCUR'10, le workshop ouvert *distributed open timed systems* (DOTS) du projet ANR éponyme (cf. *infra*) ;
- En 2009, j'ai coorganisé avec Olivier H. Roux, à Nantes, le 7e *colloque francophone sur la modélisation des systèmes réactifs* (MSR'09), `http://msr09.irccyn.ec-nantes.fr/index.php`. Nous en avons coédité les actes [E1].

1.7.4 Contrats

- Je suis le *porteur* du projet ANR ImpRo (Implementability and Robustness of Timed Systems) qui étudie l'implantation en pratique des modèles formels pour les systèmes

embarqués communicants.
  - Programme : ANR « Blanc » ;
  - Période : 2010–2013
  - Partenaires (resp. locaux) : IRCCyN (D. Lime), IRISA Rennes (C. Jard), LIP6 Paris (B. Bérard), LSV (S. Haddad) ;
  - Coordonnateur : Didier Lime ;
  - Budget : environ 382 k€ dont 120 k€ pour l'IRCCyN ;
  - Labels Pôles de Compétitivité : IDforCar et System@tic.
- J'ai été le *responsable scientifique* local, pour l'IRCCyN, du projet ANR DOTS (Distributed Open Timed Systems) sur la modélisation formelle et l'analyse de systèmes complexes.
  - Programme : ANR « Sécurité et Informatique » ;
  - Période : 2007–2011
  - Partenaires (resp. locaux) : LSV Cachan (F. Laroussinie), LaBRI Bordeaux (I. Walukiewicz), IRCCyN (D. Lime), IRISA Rennes (C. Jard), LAMSADE Paris (B. Bérard) ;
  - Coordonnateur : François Laroussinie ;
  - Budget : environ 600 k€ dont 100 k€ pour l'IRCCyN.

Chapitre 2

Synthèse de mes travaux

2.1 Introduction

2.1.1 Les modèles formels

Mes travaux s'inscrivent dans le domaine de la conception de systèmes complexes, en particulier ceux pour lesquels les contraintes temporelles revêtent une importance particulière. Les systèmes embarqués communicants (SEC) sont un exemple très important de tels systèmes : ils sont répartis, soumis à des contraintes temporelles fortes et fonctionnent avec des ressources limitées ce qui fait de leur conception une gageure. Nous pourrions également citer comme exemples des systèmes logistiques ou encore des systèmes biologiques tels que les réseaux d'interaction de gènes. Néanmoins, les SEC constituent l'application de référence de mes travaux, en particulier lorsque, couplés à un procédé qu'ils pilotent, ils sont soumis à des contraintes temporelles sur leur vitesse de réaction aux évolutions de ce procédé. On parle alors de systèmes *temps réel*.

Pour réussir à appréhender la complexité d'un tel système, les interactions entre composants, soumises à l'allocation de ressources critiques, et les contraintes temporelles, il semble indispensable de s'appuyer sur un ou plusieurs modèles qui permettent d'abstraire certaines informations et de se concentrer sur celles qui sont pertinentes. L'objectif étant ensuite de raisonner sur ces modèles, en explorant leurs différents états possibles (l'« espace d'états »), il faut utiliser un formalisme de modélisation défini mathématiquement. Dans mes travaux, les formalismes de référence sont les automates finis et les réseaux de Petri, et plus spécifiquement leurs extensions temporisées (et hybrides). Les automates sont la brique de base de la modélisation séquentielle de systèmes à événements discrets, alors que les réseaux de Petri sont un modèle naturel pour les aspects relatifs à la concurrence des événements.

2.1.2 Les modèles temporisés

L'introduction du temps dans un modèle formel permet de le rendre plus réaliste en restreignant ses comportements en fonctions de certaines contraintes temporelles et également d'exprimer et vérifier des propriétés quantitatives : par exemple « l'alarme se déclenche toujours moins de 3 s après l'occurrence d'une panne ».

Le temps dans le modèle peut être considéré sur un domaine dense ou sur un domaine discret. Il est probablement plus « naturel » de considérer le temps comme une grandeur continue. Pour la modélisation de certaines systèmes (biologiques par exemple), un domaine

dense semble donc s'imposer. Lorsque l'on modélise des systèmes informatiques par contre, le choix n'est pas si clair : les programmes agissent à des instants discrets cadencés par des horloges matérielles discrètes et leurs actions et observations sur l'environnement physique continu se font également uniquement à des instants discrets. Un domaine discret pour le temps est donc tout à fait acceptable.

Outre le réalisme, le choix d'un temps discret ou dense a un impact important sur les méthodes de calcul et de représentation de l'espace d'états.

Mes travaux concernent principalement les automates temporisés et les réseaux de Petri temporels qui sont des extensions temporelles, respectivement, des automates finis et des réseaux de Petri.

Les automates temporisés Un *automate temporisé* (*Timed Automaton*, TA) [3] est un automate fini auquel on a ajouté un ensemble fini d'horloges. Ces horloges sont des variables dont les valeurs réelles positives augmentent de façon synchrone avec le temps. Le franchissement d'un transition peut être soumis à la vérification d'une contrainte sur les horloges appelée *garde*. Il peut également être accompagné d'une remise à zéro d'un ensemble d'horloges. Nous considérons par ailleurs le modèle étendu avec des *invariants* [50], qui contraignent les valeurs d'horloges pour lesquelles on peut rester dans les localités. Ces invariants permettent de modéliser l'urgence de certaines actions.

Les réseaux de Petri temporels Les extensions temporelles des réseaux de Petri sont nombreuses (voir p.ex. [16]). Les travaux présentés ici se concentrent sur le modèle de Merlin [60], les *réseau de Petri temporels* (*Time Petri Nets*, TPN). Dans ce formalisme, on ajoute à chaque transition un intervalle de $\mathbb{R}$ à bornes rationelles positives, appelé *intervalle statique de tir*. Cet intervalle contraint la durée pendant laquelle la transition doit être sensibilisée afin d'être tirée. La notion d'urgence est également présente dans ce modèle puisque la durée de sensibilisation des transitions ne peut dépasser la borne maximale de l'intervalle (lorsque celle-ci est finie). Lorsqu'il existe une constante finie qui majore le nombre de jetons dans chaque place à tout moment, on dit que le réseau est borné. Les travaux présentés ici considèrent quasiment exclusivement des réseaux bornés car la plupart des problèmes non-triviaux sont indécidables dans le cas non borné.

2.1.3 L'analyse des modèles formels

Le « fer de lance » de l'approche formelle des systèmes est la technique du *model-checking*, introduite par Edmund M. Clarke et Allen E. Emerson, et par Joseph Sifakis au début des années 80, et pour laquelle ils reçurent le prix Turing en 2007.

Dans cette approche, on cherche à savoir si un système S vérifie une propriété ϕ. On construit alors un modèle formel M_S de S et un modèle formel de la propriété M_ϕ, en utilisant par exemple une logique temporelle (voir p. ex. [61, 29, 2]), et on vérifie automatiquement si M_S vérifie bien M_ϕ, ce que l'on note habituellement $M_S \models M_\phi$.

La réponse à ce processus est « oui » ou « non » mais, dans certains cas, on peut obtenir un témoin si la propriété est vraie, ou un contre-exemple si elle ne l'est pas, qui donnent un peu plus d'information sur le (modèle du) système et facilitent ainsi le processus de conception.

La représentativité des modèles vis-à-vis de la réalité des systèmes est un point délicat, d'autant plus que les systèmes sont généralement spécifiés de manière informelle ou incomplète : par exemple, la spécification d'un système informatique est souvent écrite en langage

naturel ou à l'aide d'un formalisme comme UML (*Unified Modeling Language*) présentant des ambiguités sémantiques.

Pour modéliser en l'absence de certaines informations, on peut introduire un ensemble de *paramètres* P dans le modèle. Ces paramètres représentent des grandeurs inconnues, par exemple un délai de communication entre deux nœuds d'un réseau. Le problème du *model-checking* se décline alors en deux variantes : existe-t-il une valeur des paramètres de P telle que $M_S \models M_\phi$? Ce problème, souvent appelé « problème du vide » (sous-entendu de l'ensemble des valeurs de paramètres assurant la vérification de la propriété, voir p.ex. [4]), apporte, comme précédemment une réponse « oui » ou « non ». Plus intéressante pour le processus de conception, le deuxième problème, dit de « synthèse de paramètres », demande de calculer l'ensemble des valeurs de paramètres telles $M_S \models M_\phi$. Ce problème est évidemment plus général, et donc plus difficile à résoudre, que celui du vide mais les informations qu'il fournit peuvent être très utiles pour le processus de conception.

L'approche du *model-checking* suppose que l'on a modélisé le système ainsi que son interaction, fixée, avec l'environnement : on vérifie donc des propriétés sur un modèle en « boucle fermée ». De façon peut-être un peu plus ambitieuse, on peut laisser libre cette interaction avec l'environnement et se demander si l'on peut adjoindre au système « ouvert » ainsi modélisé un « contrôleur » C tel que le système composé $C||M_S$ vérifie la propriété ϕ. Le contrôleur ne peut agir que sur certaines actions du système, dites « contrôlables » (ou encore commandables), et doit réagir aux autres actions, dites « incontrôlables » (non commandables), afin de garantir la satisfaction de la propriété [62]. Le « problème du contrôle » s'intéresse à l'existence d'un tel contrôleur, celui de la « synthèse de contrôleur » à sa construction effective. Le problème du contrôle est plus général que le *model-checking* dans le sens où, sous certaines conditions, ce dernier peut être représenté par un problème de contrôle où aucune action n'est contrôlable.

2.1.4 Explosion combinatoire

La principale difficulté à laquelle se heurtent les techniques d'analyse brièvement décrites dans la section précédente est connue sous le nom d'« explosion combinatoire » : la taille de l'espace d'états d'un modèle grandit parfois beaucoup plus vite que la taille de ce modèle.

Un facteur important de ce phénomène est la description d'un système en composants parallèles. L'état global du système est donné par l'état de chacun de ses composants. De cefait, si chaque composant possède au moins n états, le nombre d'états globaux d'un système à k composants indépendants est au moins n^k. Les dépendances entre composants réduisent ce nombre mais en contrepartie rendent plus compliquée l'analyse globale : des composants complètement indépendants les uns des autres peuvent être vérifiés en isolation. Plusieurs techniques peuvent être mises en œuvre pour atténuer cette explosion ou tout au moins ses effets. Citons l'utilisation de méthodes symboliques de représentation et de manipulation des états à base de diagrammes de décision (p.ex. [22, 30]), ainsi que les techniques d'exploration basées sur l'exploitation la concurrence comme les dépliages [59, 41].

Pour les modèles temporisés, on observe un problème similaire dû à l'utilisation d'horloges pour modéliser l'écoulement du temps et les durées de certains processus. Si l'adjonction de contraintes temporelles réduit le nombre de comportements discrets du modèle, elle augmente également grandement le nombre d'états du modèle. En effet, à un état du modèle sans contraintes temporelles correspond un nombre exponentiellement grand (en fonction du nombre d'horloges et des constantes numériques impliquées dans les contraintes) d'états du modèle temporisé qui décrivent également les valeurs des horloges. Pour modérer cette explo-

sion du nombre d'états, on a recours à des abstractions regroupant le maximum possible de ces états par une relation d'équivalence préservant les propriétés à vérifier (voir p. ex. [9]).

2.2 Synthèse des travaux

La plupart de mes travaux présentés dans cette section ont impliqué mes collègues nantais Franck Cassez ou Olivier H. Roux. Afin de ne pas surcharger la présentation, je ne mentionnerai à chaque fois que les collaborations extérieures. On pourra se référer aux publications associées pour retrouver leurs participations. À noter que dans [J6], il ne s'agit pas d'Olivier H. Roux mais bien de son homonyme Olivier F. Roux.

2.2.1 Positionnement et originalité

Mes travaux s'inscrivent dans le contexte général présenté dans la section précédente et concernent principalement les modèles temporisés. Les thèmes abordés sont, notamment, l'expressivité de ces modèles et leurs extensions avec des chronomètres, le *model-checking*, le contrôle au travers des jeux temporisés, les dépliages et les aspects paramétrés.

Même si plusieurs de ces contributions concernent les automates temporisés, l'objet d'étude principal de ces travaux reste les réseaux de Petri temporels.

On peut ainsi dire que globalement ces travaux se trouvent à l'interface entre trois communautés : réseaux de Petri, méthodes formelles et systèmes temps réels.

Par ailleurs, une originalité importante de mes travaux se trouve dans la mise en œuvre, autant que possible, des techniques d'analyse développées dans des (prototypes d') outils logiciels.

2.2.2 Expressivité des modèles

Contexte Bien que commençant beaucoup plus tôt, la littérature sur les réseaux de Petri temporels n'est pas très fournie comparativement à celle sur les automates temporisés. Cependant, il est clair que ces deux modèles sont très proches. Ainsi une part importante de mes travaux a consisté à étudier l'expressivité comparée de ces deux modèles dans un double objectif : situer plus finement le modèle des réseaux de Petri temporels dans la hierarchie des modèles formels temporisés, et transférer vers ces réseaux des résultats théoriques et des techniques d'analyse développés pour les automates temporisés.

Nous éliminons d'emblée le cas des réseaux de Petri temporels non bornés qui ont le pouvoir d'expression des machines de Turing [54] et nous concentrons sur les réseaux bornés.

Contributions Dans [J7, C22] nous proposons la construction d'un automate temporisé à partir du graphe des classes d'un réseau de Petri temporel borné. Cet automate est temporellement (fortement) bisimilaire au réseau de Petri et nous prouvons ainsi la décidabilité, pour les réseaux de Petri temporels, de tous les problèmes décidables pour les automates temporisés qui sont préservés par la bisimulation temporelle forte. En particulier, nous montrons également comment vérifier une variante de la logique TCTL (Timed Computation Tree Logic) dédiée aux réseaux de Petri temporels grâce à cette transformation.

Dans [C16] et [J4, C14], travaux réalisés en collaboration avec Béatrice Bérard (LIP6) et Serge Haddad (LSV), nous nous intéressons à la traduction inverse, c'est-à-dire des automates temporisés vers les réseaux de Petri temporels. Nous exhibons d'abord un automate temporisé

pour lequel nous prouvons qu'il n'existe aucun réseau de Petri temporel borné qui lui soit temporellement bisimilaire (même faiblement). Avec les résultats précédents, on en déduit que, du point de vue de la bisimulation temporelle, les réseaux de Petri temporels bornés décrivent une classe de modèle strictement plus petite que celle décrite par les automates temporisés.

Nous nous intéressons alors à la sous-classe des automates temporisés correspondant aux réseaux de Petri temporels (pour la bisimulation temporelle). [C16] en donne une caractérisation syntaxique (contraintes larges et invariants « croissants ») pour des réseaux à contraintes larges, et [J4, C14] en donne une caractérisation sémantique.

Par ailleurs, nous montrons dans [C16] que, du point de vue des langages temporisés, les automates temporisés et les réseaux de Petri temporels bornés sont équivalents. Pour obtenir ce résultat nous proposons une traduction de tout automate temporisé en un réseau de Petri temporel sauf (au plus un jeton par place) préservant le langage temporisé. Cela nous permet notamment d'établir un certains nombre de résultats négatifs concernant le langage temporisé des réseaux de Petri temporels bornés : indécidabilités du langage universel et de l'inclusion par exemple.

Une autre conséquence intéressante de ce résultat, couplé avec la traduction de [J7] par exemple, est que les réseaux de Petri temporels bornés ont la même expressivité, en termes de langages temporisés, que les réseaux de Petri temporels saufs.

Dans une extension récente des travaux de [C16], nous avons généralisé ce résultat et montré que les réseaux des Petri temporels bornés avec arcs de lecture, arcs inhibiteurs (logiques), arc de remise-à-zéro et virtuellement toutes les extensions similaires agissant uniquement sur le marquage sont équivalents aux réseaux saufs.

On voit ainsi que ces traductions nous permettent d'établir des résultats propres au modèle réseau de Petri lui-même (le caractère borné est spécifique à ce modèle). Nous nous sommes intéressés à des propriétés spécifiques également hors du contexte des liens avec les automates temporisés, comme un aspect de la politique de mise à zéro des horloges associées aux transitions lors de leur tir (la « politique de mémoire »), que nous avons étudiée dans [C15], toujours en collaboration avec Béatrice Bérard et Serge Haddad. Nous avons ainsi notamment montré qu'en relâchant certaines contraintes de la sémantique habituelle, dite « intermédiaire », on peut obtenir une sémantique strictement plus expressive du point de vue de la bisimulation temporelle faible, en présence d'intervalles ouverts. Cette sémantique, appelée « atomique persistante » préserve les résultats de décidabilité classiques. Notons que si les intervalles temporels considérés sont toujours fermés à droite, les deux sémantiques sont équivalentes.

Perspectives De nombreux problèmes intéressants restent ouverts concernant l'expressivité des réseaux de Petri temporels. Citons par exemple, l'existence de sous-classes déterminisables, analogues des *event-clock automata*, ou l'étude des langages temporels générés par les réseaux de Petri temporels en fonction de l'emplacement de l'intervalle de tir (arcs, place, transition) [1]. La comparaison de ces modèles dans le cas non borné serait également intéressante.

2.2.3 Réseaux de Petri à chronomètres

Contexte Dans les modèles temporisés, la mesure du temps se fait à l'aide d'horloges explicites ou implicites dont la valeur augmente avec le temps. La seule autre opération agissant

1. [16] étudie ce problème du point de vue de la bisimulation faible.

sur les valeurs de ces horloges est la mise à zéro.

Dans le cadre des systèmes temps réels, on peut notamment utiliser ces horloges pour modéliser le temps d'exécution d'une tâche. Cependant, dans un système multitâches ordonnancé avec une politique préemptive, une tâche peut être interrompue par une autre, plus prioritaire, puis reprise plus tard sans avoir à recommencer de zéro. Pour modéliser cela avec un modèle temporisé, il nous faut la notion de chronomètre, horloge qui peut être « gelée » pendant un certain temps, gardant cependant sa valeur, puis redémarrée un peu plus tard à sa valeur précédente. L'introduction de ces chronomètres a été étudiée dans le cadre des modèles basés sur les automates, notamment dans [49, 25] mais au moment de nos travaux, rien n'existait dans le cadre des réseaux de Petri temporels.

Contributions Nous avons étudié l'impact de l'introduction de ces chronomètres dans les réseaux de Petri temporels à plusieurs niveaux : du point de vue de l'adéquation du modèle pour la modélisation de systèmes temps réels à ordonnancement préemptif [J3, C23] : comment modéliser des tâches communicantes (sémaphores, événements, etc.) en présence d'un algorithme d'ordonnancement à priorités fixes avec accès aux ressources (protocoles de priorité héritée). Nous avons également montré comment modéliser d'autres politiques d'ordonnancement préemptives : le temps partagé (*Round-Robin*) et un algorithme à priorités dynamiques (*Earliest Deadline First*). Pour le temps partagé, nous avons introduit des variables de temps à dérivées constantes dans le même esprit que les automates hybrides [49].

Nous avons proposé plusieurs possibilités syntaxiques pour l'introduction des chronomètres. En effet, dans les réseaux de Petri temporels les variables continues sont implicites à la différence, par exemple, des automates temporisés pour lesquels les horloges sont explicites. Nous avons proposé de spécifier indirectement les chronomètres à l'aide d'informations spécifiques à l'ordonnancement ajoutées aux places du réseau [J3, C23], d'arcs inhibiteurs *temporels* [C21] et finalement d'arcs *activateurs* [J5]. Notons également un travail parallèle de Bucci et co-auteurs dérivant les chronomètres d'informations spécifiques d'ordonnancement associées aux transitions du réseau [19].

Nous avons également abordé le point de vue du calcul de l'espace d'états de ces modèles dans [J3, C21] en proposant une extension du calcul classique du graphe des classes d'états pour ces modèles à chronomètres et hybrides. Cette méthode de calcul, basée sur les variables continues et non sur la façon dont ces variables sont obtenues, est valable pour toutes les variantes présentées ci-dessus. Nous avons également étendu dans [J3, C20] la construction de [J7] pour obtenir, sous réserve de terminaison, un automate à chronomètres (resp. hybride) temporellement bisimilaire à un réseau de Petri à chronomètres (resp. hybride dans le cas du temps partagé) borné donné.

Ce calcul de l'espace d'états d'un réseau de Petri à chronomètres nécessite l'utilisation de polyèdres généraux, au lieu de « zones » (voir le chapitre 3) pour les réseaux de Petri temporels, et la manipulation de ces polyèdres est très coûteuse en temps de calcul et en mémoire. En remarquant que, malgré tout, le polyèdre initial toujours est une zone, nous avons prouvé une condition syntaxique nécessaire et suffisante sur une zone pour que son successeur ne soit pas une zone. Déterminer si un polyèdre est une zone étant très facile, nous avons donc obtenu une amélioration heuristique du semi-algorithme de calcul de l'espace d'états utilisant des zones, et uniquement en cas de besoin, des polyèdres [C19].

Nous avons aussi montré dans [J5], en collaboration avec Bernard Berthomieu et François Vernadat (LAAS), que, comme pour les automates hybrides, on peut simuler une machine

de Minsky à deux compteurs à l'aide d'un réseau de Petri à chronomètres, même borné, prouvant ainsi l'indécidabilité de la plupart des problèmes intéressants tels que l'accessibilité de marquages.

Pour contourner ce problème nous avons proposé des surapproximations finies de l'espace d'états, à base de zones (p. ex. dans [C21]) ou de polyèdres reposants sur une grille de pas paramétrable [J5].

Nous avons également étudié l'impact d'une sémantique en temps discret pour les réseaux de Petri à chronomètres puis montré dans ce cas la décidabilité des problèmes intéressants et montré comment calculer l'espace d'états de manière symbolique [C11] (voir la section 2.2.6).

Plus récemment, nous avons étendu la notion de dépliage symbolique (voir la section 2.2.5 et le chapitre 4) pour les réseaux de Petri à chronomètres [C6] et montré leur utilisation, même en l'absence d'un préfixe fini complet, pour la supervision [C7].

Perspectives Pour bénéficier pleinement de ces modèles à chronomètres, le problème ouvert principal reste de trouver des sous-classes très expressives. De ce point de vue, le temps discret est certainement l'une des voies les plus prometteuses et il convient d'améliorer le calcul symbolique proposé dans [C11].

Il serait également intéressant d'essayer d'« élargir » les polyèdres générés par ce calcul symbolique afin de se rapprocher de ceux générés par le calcul en temps dense (et dans lesquels ils sont inclus) tout en en gardant un nombre fini. Une voie possible pourrait être d'utiliser à nouveau des polyèdres dont les sommets se situent sur une grille de pas paramétrable, mais cette fois inclus dans l'abstraction en temps dense.

2.2.4 Jeux temporisés

Contexte Les bases théoriques du contrôle pour les modèles formels temporisés (formulés comme des jeux temporisés) ont été posées par Oded Maler, Amir Pnueli, Eugène Asarin et Joseph Sifakis à la fin des années 90 [58, 5]. Ces résultats étendent ceux du cas discret et proposent un calcul des états contrôlables par itération du calcul de *prédécesseurs contrôlables* à la fois discrets et temporels.

La principale motivation de nos travaux dans ce domaine était l'absence d'implémentation efficace de ces algorithmes. Par ailleurs, pour le problème de la vérification des modèles formels temporisés, l'outil Uppaal [56] développé conjointement par l'université d'Aalborg au Danemark et l'université d'Uppsala en Suède, est devenu une référence au fil des années. L'une des raison de ce succès est justement l'efficacité de cet outil, à laquelle contribuent notamment deux techniques clés : la représentation de l'espace d'états à l'aide de zones et l'utilisation d'algorithmes *à la volée* permettant une terminaison anticipée des calculs dans de nombreux cas. Notre objectif initial a donc été le développement d'algorithmes pour les jeux temporisés incorporant ces deux techniques et de développer un outil efficace basé sur ces algorithmes.

Contributions Dans [C17], nous avons donc proposé le premier algorithme intégralement à la volée et basé entièrement sur les zones, pour le calcul des états gagnants d'un jeu temporisé d'accessibilité, ainsi qu'une variante pour le calcul de la stratégie permettant d'atteindre une état donné le plus rapidement possible. Nous avons également développé un prototype, nommé Uppaal-TiGa, implémentant ces algorithmes et utilisant la bibliothèque de représentation et manipulation des zones d'Uppaal développée par Alexandre David [31].

Nous avons par la suite complètement intégré ce prototype dans l'environnement logiciel d'Uppaal, permettant ainsi notamment l'utilisation de la quasi-intégralité de son langage de modélisation (variables, fonctions, *templates*, etc.) et d'un certain nombre d'optimisations génériques. Cette version plus aboutie d'Uppaal-TiGa a été présentée dans [C13, W3].

Nous nous sommes également intéressés au problème des jeux temporisés d'accessibilité sous observation partielle. Si l'on considère une observabilité partielle des événements, le problème associé d'existence d'un contrôleur est indécidable. Il est 2EXPTIME-complet si le contrôleur ne peut avoir accès qu'à des ressources « bornées » (nombre maximal d'horloges et constantes maximales pour les gardes fixés *a priori*) [14]. Afin d'obtenir un algorithme nous avons considéré dans [C12] un nombre fini d'observations sur les états et sur le temps. Par une technique de « calcul de sous-ensembles » nous avons montré que ce jeu est alors équivalent à un jeu non temporisé à observation totale. Nous avons alors proposé un algorithme réalisant conjointement la construction de ce jeu et sa résolution à l'aide d'une variante de l'algorithme pour les jeux d'accessibilité d'Uppaal-TiGa. Ce nouvel algorithme ainsi que la possibilité de spécifier les problèmes d'observabilité partielle ont également été implémentés dans Uppaal-TiGa.

La disponibilité d'un outil efficace pour les jeux temporisés a encouragé leur application à la résolution d'autres problèmes théoriques et pratiques sur les systèmes temporisés, notamment par l'équipe de Kim G. Larsen à Aalborg. Citons par exemple la simulation alternante d'automates temporisés [21], la génération de tests temps réel [37, 35, 36] et une théorie d'interface pour les systèmes temps réels [32, 33, 34].

Pour cette dernière application, une version spécialisée d'Uppaal-TiGa, nommée ECDAR, a été développée. Dans le cadre de ces travaux, j'ai contribué par le développement d'une variante de l'algorithme des jeux d'accessibilité de Uppaal-TiGa permettant de résoudre des jeux de Büchi, à l'ajout de contraintes d'implémentabilité et de vivacité dans les interfaces [C3].

Perspectives On montre dans le chapitre 3 comment étendre l'algorithme de [C17] pour calculer des plus petits points fixes non-imbriqués arbitraires. Une perspective importante serait d'étendre encore cet algorithme pour calculer des plus grands points fixes et surtout pour des points fixes imbriqués de façon efficace. Si le premier point semble raisonnablement facile, le second paraît non-trivial, sauf à perdre en grande partie le caractère à-la-volée de l'algorithme.

Un autre problème intéressant mais très difficile serait de définir des jeux temporisés sur les dépliages symboliques, avec des stratégies prenant en compte la seule mémoire causale. Mais avant même d'en arriver là, le cas non temporisé reste à résoudre et semble également difficile.

2.2.5 Dépliages

Contexte Depuis les articles fondateurs de Ken McMillan [59] et Javier Esparza [41], les dépliages de systèmes concurrents ont bénéficié d'une grande variété de travaux (on pourra se référer à [42] pour une vision globale). L'extension de cette notion aux systèmes temporisés est cependant relativement récente avec des travaux sur les réseaux de Petri temporels [27] et sur les réseaux d'automates temporisés [23, 15].

Tout l'enjeu de l'introduction du temps, et surtout de l'urgence, dans la notion de dépliage consiste à capturer les causalités supplémentaires entre événements concurrents qu'elle induit.

Pour prendre en compte ces causalités, Thomas Chatain et Claude Jard ajoutent des arcs de lecture et des duplications de transitions dans les dépliages de réseaux de Petri temporels.

Contributions Dans le cadre de la thèse de Louis-Marie Traonouez, et en collaboration avec Claude Jard et Bartosz Grabiec, nous avons proposé une technique différente qui permet d'exprimer le dépliage symbolique comme un étiquetage par des dates d'événements du dépliage du réseau non temporisé sous-jacent. Les causalités dues au temps sont alors reflétées par des contraintes supplémentaires dans l'expression symbolique des dates des événements. Nous avons montré comment obtenir un préfixe complet fini du dépliage symbolique pour des réseaux de Petri temporels bornés. Nous avons également étendu cette technique pour prendre en compte des chronomètres ou des paramètres temporels [C6].

Dans le cas d'un réseau de Petri temporel non borné, ou contenant des chronomètres ou des paramètres, l'existence d'un préfixe complet *fini* n'est plus assurée. Nous avons cependant montré comment utiliser en pratique ce dépliage dans le cadre de la supervision en ligne, en construisant un dépliage partiel, guidé par les observations. Nous avons montré, pour des modèles à paramètres comment ce dépliage partiel peut être utilisé pour inférer des valeurs de paramètres expliquant les observations [C7].

Perspectives Outre les perspectives de combinaison avec les jeux évoqués précédemment, il serait intéressant de formaliser les différences des approches pour le dépliage symbolique des réseaux de Petri temporels mais également pour les réseaux d'automates temporisés. En cas de succès, on pourrait alors envisager d'unifier ces différentes approches.

D'un point de vue plus pratique, nous souhaitons également implanter notre approche de façon plus efficace, afin de la rendre vraiment concurrentielle avec les implémentations basées sur la sémantique séquentielle.

Enfin, il serait intéressant de définir un dépliage symbolique pour les réseaux de Petri stochastiques de [20].

2.2.6 Temps discret

Contexte La plupart de mes travaux concernent des modèles où le temps est représenté de façon continue. Cependant, comme nous l'avons vu dans la section 2.1.2, l'utilisation d'un temps discret peut apporter de meilleurs résultats théoriques en termes de complexité ou de décidabilité, voire, pour certaines applications, être plus proche du système modélisé.

En contrepartie, le nombre d'états des modèles en temps discret est en général très grand. En particulier, il est très sensible à la grandeur des constantes de temps du système. Ainsi un simple réseau de Petri à une transition d'intervalle [1,1000] possède déjà 1000 états différents, chiffre qui grandit exponentiellement avec le nombre de transitions de ce type sensibilisées simultanément.

Contributions Pour résoudre ce problème, nous avons exploré deux pistes : la première consiste à utiliser les algorithmes symboliques du temps continu pour traiter ces états par blocs. On les représente alors par le polyèdre constitué par leur enveloppe convexe. À chaque calcul de successeur, il convient alors d'évaluer les états discrets supplémentaires créés par cette surapproximation, et de les éliminer (ce qui conduit en général à une union finie de polyèdres). On prouve que l'algorithme symbolique obtenu en modifiant le calcul du graphe

des classes d'un réseau de Petri à chronomètres selon ce principe converge et permet de calculer exactement l'espace d'états discret [C11].

L'autre possibilité que nous avons envisagée consiste à utiliser une structure de données efficace pour représenter les grands espaces d'états. En collaboration avec Yann Thierry-Mieg, Fabrice Kordon et Béatrice Bérard du LIP6, nous avons donc étudié comment les *Hierarchical Set Decision Diagrams* (SDD) qu'ils développent peuvent être utilisés dans ce contexte. Le résultat est finalement assez naturel et un prototype a été développé. Il est interfaçable avec notre outil Roméo et se révèle extrêmement performant dans de nombreux cas [C5].

Perspectives La perspective la plus notable concernant l'utilisation du temps discret consiste en l'amélioration de l'exploration symbolique évoquée dans la section sur les chronomètres.

2.2.7 Modèles paramétrés

Contexte L'introduction de paramètres dans les gardes ou les invariants des automates temporisés conduit à des problèmes indécidables. Ainsi le problème le plus simple de l'existence de valeurs de paramètres telles qu'une certaine localité est accessible est indécidable [4].

Néanmoins, l'utilisation de paramètres présente un intérêt pratique indéniable pour la modélisation et la vérification de systèmes complexes. Comme pour les chronomètres, il est ainsi intéressant de développer des semi-algorithmes qui permettent dans certains cas pratiques de synthétiser les valeurs des paramètres permettant la satisfaction de propriétés données.

Contributions Dans cette optique, et dans le cadre de la thèse de Louis-Marie Traonouez, nous avons étudié le problème de la vérification paramétrée pour les réseaux de Petri temporels et, sans surprise, nous avons pu transférer à ce modèle les résultats d'indécidabilité disponibles pour les automates temporisés. Nous avons ensuite proposé des semi-algorithmes pour la synthèse de paramètres assurant la vérification de propriétés TCTL non imbriquées sur les réseaux de Petri temporels [J2, C10] et nous les avons implémentés dans notre outil Roméo [C9].

Comme vu dans la section sur les dépliages, nous avons, de façon similaire, étendu la notion de dépliage symbolique d'un réseau de Petri temporel pour prendre en compte l'utilisation de paramètres dans les intervalles de temps des transitions [C6]. Nous en avons réalisé une implémentation rudimentaire dans Roméo et avons appliqué cette technique à la synthèse de paramètres dans le cadre d'une technique de supervision expliquant un ensemble partiellement ordonné d'observations [C7].

Enfin, nous avons considéré des réseaux de Petri à bornes entières et étudié comment « rétrécir » certains intervalles temporels bornés pour assurer la satisfaction de propriétés TCTL non imbriquées [F1]. Le nombre de « rétrécissements » possibles est borné et ceux-ci s'expriment grâce à des paramètres à valeurs entières bornées. Le problème est donc décidable. Nous avons en outre montré comment calculer l'ensemble de ces rétrécissements de manière symbolique (plutôt que de tous les énumérer) grâce à des variantes des algorithmes de [J2, C10] dans lesquels le polyèdre représentant les contraintes entre les paramètres et les horloges est « corrigé », en en prenant l'enveloppe entière, à chaque étape afin que ses sommets restent entiers. Ce calcul symbolique a un double avantage par rapport à une énumération de toutes les valeurs possibles : il permet de diminuer la sensibilité du calcul à la taille de l'espace des valeurs des paramètres et fournit directement comme résultat un ensemble de contraintes linéaires entre les paramètres.

Dans le cadre de la thèse d'Aleksandra Jovanović nous avons très récemment généralisé cette idée pour la synthèse générale de paramètres entiers bornés pour les automates temporisés. Les restrictions syntaxiques imposées par les automates L/U [52] pour obtenir la décidabilité de l'existence de valeurs de paramètres pour les problèmes d'accessibilités ne sont pas satisfaisantes : nous avons montré qu'elles ne permettent pas d'assurer la décidabilités pour des propriétés proches comme l'inévitabilité et l'existence d'algorithme de synthèse de l'ensemble des valeurs de paramètres assurant la propriété n'est pour l'instant assurée que pour des sous-classes encore plus restreintes [18] dont l'utilisabilité en pratique n'est pas claire. Limiter les valeurs possibles des paramètres aux entiers bornés semble donc une alternative convaincante. Nous avons montré que l'existence de valeurs de paramètres assurant l'accessibilité ou l'inévitabilité était EXPTIME-complet et nous avons donné des algorithmes symboliques pour calculer l'ensemble de ces valeurs que nous avons implantés dans Roméo, en les appliquant au cas des réseaux de Petri temporels paramétrés.

Toujours dans le cadre de la thèse d'Aleksandra Jovanović, nous avons très récemment proposé une « paramétrisation » de l'algorithme de [C17] afin de pouvoir analyser les jeux temporisés paramétrés. L'objectif final, qui s'inscrit dans le projet ANR ImpRo, étant d'utiliser des jeux pour modéliser des relations de simulation entre un modèle d'un système et un modèle de son implémentation en pratique dans lequel les paramètres représentent l'incertitude sur le comportement de l'implémentation.

Perspectives Nous souhaitons étendre les résultats sur la synthèse de paramètres entiers aux cas des jeux temporisés et des automates hybrides linéaires. Dans ce dernier cas, il semble nécessaire de considérer des restrictions sur les variables dont les dynamiques peuvent changer, par exemple supposer que leur domaine de valeurs est discret.

Il reste également à évaluer la complexité des algorithmes symboliques proposés.

Enfin, nous souhaitons appliquer ces techniques pour l'analyse paramétrée des systèmes temporisés aux problèmes de robustesse d'implémentation des modèles temporisés.

2.2.8 Applications à la sécurité

Dans le cadre de la thèse de Gilles Benattar, nous nous sommes intéressés à l'application des modèles et des méthodes formels pour la sécurité. Dans un premier temps nous nous sommes intéressés à la propriété de non-interférence, puis nous avons étudié la détection de canaux cachés.

Non-interférence

Contexte La propriété de non-interférence s'exprime pour des systèmes dans lesquels interviennent deux types d'agents : des agents « normaux » et des agents privilégiés. Un système est *non-interférent* si le système dans lequel les actions des agents privilégiés sont invisibles est indiscernable du système dans lequel les agents privilégiés ne font rien [45]. Autrement dit un agent normal ne peut pas déduire d'information sur le comportement des agents provilégiés à partir de l'observation des seules actions non-privilégiées.

Nous modélisons de tels systèmes par des systèmes de transitions étiquetées par un alphabet partitionné entre les deux types d'agent. La propriété de non-interférence s'exprime alors comme une équivalence entre le système dans lequel les transitions privilégiées sont étiquetées par ϵ, l'action silencieuse, et celui dans lequel toutes ces transitions ont été supprimées. Cette

équivalence peut être l'égalité de langage, la cosimulation, la bisimulation, etc. Chacune de ces équivalences définit une variante de la propriété de non-interférence. Pour l'égalité de langage on parle de SNNI (pour *Strong non-deterministic non interference*), pour la cosimulation de CSNNI (pour *cosimulation SNNI*) et pour la bisimulation de BSNNI (pour *bisimulation SNNI*).

Contributions Dans [C8], nous nous sommes intéressés plus particulièrement au problème du contrôle de la non-interférence : trouver un contrôleur du système qui le rend non-interférent [46].

Nous avons montré que, dans le cas général, il n'existe pas de contrôleur le plus permissif (qui interdit un minimum d'actions contrôlables) pour rendre un système CSNNI ou BSNNI. Il en existe cependant pour la SNNI et nous avons donné un algorithme pour le calculer.

Nous avons également étendu ces résultats au cas temporisé, c'est-à-dire, lorsque le système est modélisé par un automate temporisé. Dans ce cas, le problème du contrôle de la SNNI est indécidable : c'est déjà le cas pour le problème de la simple vérification [26, C8]. Nous avons cependant proposé une sous-classe syntaxique d'automates temporisés déterministes sur les actions non-privilégiées pour laquelle nous avons prouvé que le problème est EXPTIME-complet et nous avons étendu notre algorithme pour calculer dans ce cas le contrôleur le plus permissif.

Perspectives La perspective principale pour ces travaux est la détermination de conditions pour lesquelles il existe un contrôleur le plus permissif pour la BSNNI et la CSNNI et la conception d'un algorithme de calcul de ce contrôleur dans ce cas.

Canaux cachés

Contexte La propriété de non-interférence, quelle que soit sa variante, est une propriété extrêmement « forte » : il suffit d'une fuite d'information de un bit pour qu'un système soit non-interférent. Dans le cadre d'une collaboration avec John Mullins, de l'École Polytechnique de Montréal et Béatrice Bérard et Mathieu Sassolas du LIP6, nous nous sommes donc intéressés à la propriété moins forte : l'existence d'un canal caché, c'est-à-dire d'une possibilité non spécifiée de communication entre l'intérieur du système et son environnement.

Contributions Dans le cas non-temporisé, nous avons modélisé ce problème à l'aide de transducteurs finis. L'existence d'un canal (caché) de communication binaire est alors équivalente à l'existence d'un transducteur encodeur de $\{0,1\}$ vers les actions privilégiées et d'un transducteur décodeur des actions normales vers $\{0,1\}$ tels que la composition de l'encodeur, du système et du décodeur soit la relation rationnelle identité. Nous avons également introduit la prise en compte de retards finis de transmission ainsi que d'un nombre fini d'erreurs de transmission.

Nous avons montré qu'étant donné un décodeur et un encodeur candidats, savoir s'ils réalisent un canal est décidable. Nous avons également montré que l'existence d'un canal implique l'existence d'encodeurs et de décodeurs d'une forme très simple [W2]. Cependant, nous avons prouvé, à l'aide d'une réduction du problème de correspondance de Post, que l'existence d'un canal dans ce cadre est un problème indécidable mais que si le système est modélisable par un transducteur fonctionnel (chaque mot en entrée a au plus une image

en sortie) alors le problème est décidable [C4, J1]. Ce dernier résultat s'étend au cas de transducteurs de norme finie, c'est-à-dire à l'union finie de transducteurs fonctionnels [10].

Perspectives Un premier problème intéressant à résoudre consiste à trouver une sous-classe décidable plus générale que les transducteurs à norme finie. Les transducteurs déterministes semblent un bon candidat pour cela.

On pourrait également étendre ces résultats aux cas temporisés ou probabilistes.

Enfin, pour aller plus loin que la simple détection, on pourrait se poser, comme pour la non-interférence ci-dessus, le problème du contrôle : existe-t-il un contrôleur tel que le système contrôlé ne contient aucun canal caché ?

2.2.9 Roméo

En parallèle de nos développements théoriques, nous essayons dans la mesure du possible d'implémenter nos algorithmes ou semi-algorithmes au sein d'un outil logiciel prototype : Roméo. Les fonctionnalités couvertes sont les traductions des TPN vers les TA, le model-checking et le calcul de l'espace d'états pour les TPN, y compris avec chronomètres et paramètres, et une implémentation préliminaire de la supervision utilisant les dépliages de TPN. Il est disponible sous license CeCILL à l'adresse `http://romeo.rts-software.org`.

Ce logiciel consiste en une interface graphique d'environ 13000 lignes de Tcl/Tk et un moteur de calcul d'environ 35000 lignes de C++ qui utilise, pour la représentation symbolique des états la bibliothèque de manipulation de DBM d'Uppaal [31] et la bibliothèque de manipulation de polyèdres *Parma Polyhedra Library* [7].

Je suis le principal auteur de ce module de calcul, avec des apports importants de Guillaume Gardey pendant sa thèse (2002-2005), et j'ai supervisé depuis 2005 les contributions ponctuelles de doctorants, les plus notables étant celles de Louis-Marie Traonouez concernant l'introduction des paramètres et les dépliages.

2.3 Conclusion

S'il faut hiérarchiser un peu ces différents résultats, il me semble que mes contributions les plus importantes et qui ont actuellement le plus d'impact sur la communauté sont sur le contrôle des systèmes temporisés et l'introduction des chronomètres dans les réseaux de Petri.

D'une part, nous avons rendu accessibles en pratique les techniques de résolution des jeux temporisés en proposant de nouveaux algorithmes à-la-volée et reposant sur l'utilisation des zones, ce qui a permis de les intégrer dans l'écosystème d'Uppaal et d'obtenir l'outil Uppaal-TiGa qui est à la fois efficace et convivial. Nous avons également proposé un cadre pour les jeux temporisés sous observation partielle qui est également implémenté dans Uppaal-TiGa. L'influence de ces travaux est notamment perceptible au travers leur utilisation dans plusieurs cas d'études, et aussi, comme nous l'avons vu plus haut, comme base de plusieurs développements théoriques sur le test, la simulation d'automates temporisés ou les théories d'interface pour la spécification des systèmes temporisés.

D'autre part, nous avons étendu les réseaux de Petri avec des chronomètres et étudié leur applicabilité pour la modélisation des systèmes temps réels avec ordonnancements préemptifs. Malgré une expressivité légèrement inférieure des TPN bornés vis-à-vis des TA, nous avons montré que l'ajout de chronomètres aux TPN bornés rend également l'accessibilité indécidable.

Nous avons proposé deux techniques de surapproximation de l'espace d'états permettant la terminaison dans certains cas. Nous avons également étudié finement les cas où les zones représentant les états symboliques ne sont plus suffisantes et proposé un algorithme permettant d'utiliser les polyèdres généraux uniquement lorsque nécessaire. Pour obtenir un cadre décidable, nous avons considéré une sémantique en temps discret et nous avons montré comment calculer l'espace d'état de façon symbolique, sans avoir à énumerer tous les états. Enfin, nous avons défini la notion de dépliage symbolique pour les réseaux de Petri à chronomètres. La plupart de ces travaux ont fait l'objet d'une implémentation dans Roméo.

Il me semble par ailleurs que les résultats que nous avons obtenus sur le dépliage des réseaux de Petri temporels ont également un certain potentiel. Nous avons montré comment calculer un dépliage symbolique pour des réseaux de Petri temporels, mais également en présence de chronomètres et de paramètres. Dans le premier cas, nous avons prouvé l'existence d'un préfixe complet fini. Pour les deux autres cas, nous avons adapté une technique d'analyse par supervision où une observation contraint le dépliage et le rend fini. Outre les perspectives évoquées plus haut, il me semble qu'il serait important de proposer des algorithmes efficaces pour calculer ces dépliages temporisés, et d'obtenir ainsi une implémentation compétitive avec celles disponibles pour les approches séquentielles.

Enfin, mes contributions récentes sur les aspects paramétrés, en particulier la recherche de paramètres entiers bornés, sont, je pense, assez originaux et d'une portée pratique indéniable. Ces résultats sont déjà mis en œuvre dans Roméo, et malgré une complexité algorithmique importante, la garantie de terminaison devrait permettre d'appliquer l'analyse paramétré à un nombre plus important de systèmes. Par ailleurs, la technique employée (utilisation de l'enveloppe entière des polyèdres) est vraisemblablement suffisamment générique pour s'appliquer assez directement à de nombreux algorithmes d'analyse, en particulier le contrôle et les dépliages paramétrés.

Parmi toutes les perspectives évoquées dans cette synthèse, le court terme consiste principalement en des travaux sur les dépliages et les aspects paramétrés, notammment leur utilisation dans le cadre du contrôle, mais aussi sur la sémantique discrète qui semble prometteuse pour l'application des techniques de recherche de paramètre entiers aux modèles à chronomètres.

Un objectif particulier est d'utiliser les modèles paramétrés pour résoudre des problèmes de robustesse d'implémentabilité dans le cadre du projet ANR BLANC ImpRo que je coordonne : avec un jeu temporisé paramétré on peut modéliser, grâce aux paramètres, une part des incertitudes temporelles introduites par une implémentation en pratique, puis trouver les valeurs de ces paramètres telles que, par exemple, le modèle de l'implémentation soit bisimilaire (en temps abstrait) au modèle du système de départ. La sémantique discrète est également centrale pour les problèmes de robustesse d'implémentation puisque les calculateurs fonctionnent, *in fine*, en temps discret.

À plus long terme, toutes les perspectives présentées sont intéressantes, et nous en aborderons la plupart, mais il me semble notamment important d'aller vers des modèles probabilistes qui permettront des modélisations plus précises dans certains cas et complèteront les fonctionnalités de Roméo. Les premiers pas dans cette direction pourront être la définition d'un dépliage symbolique pour les réseaux de Petri stochastiques et l'étude de sous-classes, par exemple la restriction à des lois normales pour les distributions temporelles associées aux transitions.

Deuxième partie

Exposés techniques détaillés

Introduction

Dans cette partie, je présente de manière détaillée des contributions représentatives de mon activité de recherche et de mon positionnement dans la communauté scientifique.

Ainsi, les deux ensembles de contributions présentés s'intéressent-ils tous deux à l'analyse des systèmes temporisés et incluent des aspects algorithmiques mis en œuvre dans des outils (avec divers niveaux de maturité).

Le chapitre 3 aborde le problème du contrôle, au travers du formalisme des jeux temporisés, et le chapitre 4 s'intéresse aux aspects liés à la concurrence, par la construction d'un dépliage symbolique de réseaux de Petri temporels.

Ces deux ensembles de contributions s'intègrent naturellement dans le projet ANR SETI 2006 *Distributed Open Timed Systems (DOTS)* dont j'ai été le responsable local pour l'IRCCyN de 2007 à 2011 : ils font en effet partie de façon évidente les axes « temporisé et contrôle » et » temporisé et distribué » du projet.

Chapitre 3

Jeux temporisés

3.1 Introduction

3.1.1 Objectifs

L'objectif de ces travaux est de produire un outil « efficace » pour l'analyse des jeux temporisés et notamment pour le contrôle des systèmes temporisés.

La décidabilité de ces jeux temporisée est connue depuis relativement longtemps [58, 5, 39] mais ces résultats, basés sur le calcul des états gagnants par le calcul d'un point fixe en arrière ne permettent pas une implémentation efficace, comme nous le verrons dans la section 3.4.2. Les travaux présentés dans [65, 1] offrent un algorithme à la volée pour résoudre les jeux temporisés d'accessibilité mais celui-ci requiert le calcul préalable d'un graphe quotient vis-à-vis de la bisimulation temps abstrait (*time-abstract bisimulation*) du système de transitions temporisé représentant la sémantique du jeu. Ce calcul est évidemment très coûteux.

Néanmoins, l'approche « à la volée » permet l'évaluation de propriétés du domaine de façon locale et sans calculer nécessairement tout l'espace d'états du modèle. Elle est au cœur de l'efficacité des algorithmes implémentés par des outils de vérification comme SPIN [51] pour les systèmes finis ou encore Kronos [17] et Uppaal [56] pour les systèmes temporisés.

Notre objectif est donc la production d'algorithmes à la volée et ne calculant que des états effectivement accessibles. Dans un premier temps, nous nous concentrons sur les jeux d'accessibilité (section 3.4) puis nous montrons comment adapter l'algorithme proposé pour les jeux d'accessibilité en temps optimal, les jeux de sûreté et les jeux de Büchi (section 3.5).

Nous discutons aussi également le traitement de l'urgence et des invariants dans les jeux temporisés (section 3.3).

3.1.2 Contexte

Ces travaux reposent principalement sur une collaboration avec Kim Larsen et Alexandre David de l'Université d'Aalborg. Ils ont démarré lors de mon postdoctorat à l'université d'Aalborg, dont les principaux résultats ont été l'algorithme SOTFTR et le premier prototype d'Uppaal-Tiga [C17].

Par la suite, nous nous sommes intéressés au contrôle sous observation partielle [C12] et plus récemment à l'utilisation des objectifs de Büchi pour la spécification de systèmes temporisés au sein d'une théorie d'interfaces [C3].

En parallèle de ces aspects théoriques, l'implémentation d'UPPAAL-TIGA a grandement progressé, avec notamment une intégration complète dans l'environnement d'UPPAAL permettant de bénéficier de toute la richesse du formalisme de modélisation ainsi que de nombreuses optimisations génériques [W3, C13]. Un certain nombre de ces développements ont été réalisés par des stagiaires de l'École Centrale de Nantes encadrés par Alexandre David et suivis par moi dans le cadre de leur stage de fin d'études.

Les résultats présentés dans ce chapitre ont été en partie publiés dans [C17],[C13] et [C3]. Les résultats de [C17], en particulier, sont présentés sous une forme étendue. Les preuves pour les résultats nouveaux ou étendus sont données dans l'annexe A.

3.2 Définitions

3.2.1 Notations et définitions préliminaires

Nous notons $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$ respectivement les ensembles de nombres entiers, rationnels et réels. $\mathbb{R}_{\geq 0}$ (resp. $\mathbb{Q}_{\geq 0}$) désigne l'ensemble des réels (resp. rationnels) positifs ou nuls. $\mathbb{Q}_{>0} = \mathbb{Q}_{\geq 0} \backslash \{0\}$ et $\mathbb{R}_{>0} = \mathbb{R}_{\geq 0} \setminus \{0\}$. $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ est l'ensemble des booléens.

Nous notons $f : A \to B$ pour indiquer que le domaine d'une fonction f est A et que son codomaine est B. Pour un ensemble A, $|A|$ désigne le nombre d'éléments de A si celui-ci est fini, et vaut $+\infty$ sinon. $\overline{A}$ désigne le complémentaire de A. Quand A est fini, nous notons 2^A l'ensemble des parties de A.

Pour un ensemble fini de variables X, nous notons $\mathcal{Z}(X)$ l'ensemble des contraintes φ générées par la grammaire :

$$\varphi ::= \mathsf{true} | x \sim k | x - y \sim k | \varphi \wedge \varphi$$

avec $k \in \mathbb{Q}$, x,$y \in X$ et $\sim \in \{<, \leq, =, \geq, >\}$.

$\mathcal{C}(X)$ est le sous-ensemble de $\mathcal{Z}(X)$ généré par la grammaire $\varphi ::= \mathsf{true} | x \sim k | \varphi \wedge \varphi$. $\mathcal{B}(X)$ est le sous-ensemble de $\mathcal{C}(X)$ généré par la grammaire $\varphi ::= \mathsf{true} | x \leq k | x < k | \varphi \wedge \varphi$.

Une valuation réelle (positive) sur X est une fonction de X dans $\mathbb{R}_{\geq 0}$. Nous notons $\mathbb{R}_{\geq 0}^X$ l'ensemble des valuations réelles et $\mathbf{0}_X$ la valuation nulle telle que $\forall x \in X, \mathbf{0}_X(x) = 0$. Étant donnés une valuation v et un sous-ensemble Y de X, nous notons $v[Y]$ la valuation telle que $v[Y](x) = 0$ si $x \in Y$ et $v[Y](x) = v(x)$ si $x \notin Y$. Pour $d \in \mathbb{R}_{\geq 0}$, nous notons $v + t$ la valuation telle que $\forall x \in X, (v + d)(x) = v(x) + d$. Soit g une contrainte de $\mathcal{Z}(X)$, nous écrivons $v \models g$ ssi l'expression obtenue en remplaçant chaque variable x dans g par $v(x)$ a pour valeur true. Pour une contrainte $g \in \mathcal{Z}(X)$, nous notons $[\![g]\!]$ l'ensemble des valuations v telles que $v \models g$.

3.2.2 Structure de jeu temporisé

Les jeux temporisés qui nous intéressent sont définis de façon finie à l'aide de *structures de jeu temporisé*. Une telle structure précise les *règles* du jeu : quelles sont les actions possibles pour chaque joueur et sous quelles conditions elles peuvent être jouées. Le formalisme que nous adoptons se base sur les automates temporisés [3, 50] : une structure de jeu temporisé est un automate temporisé dont l'alphabet des actions a été réparti, de façon disjointe, entre les deux joueurs. Cela correspond aux automates de jeu temporisé (*Timed Game Automata*) de [58, 5].

Nous donnons la définition permettant des invariants stricts (p. ex. $x < 4$) mais nous supposons pour l'instant que les invariants sont non-stricts, jusqu'à la section 3.3 où nous verrons comment lever cette hypothèse.

Définition 1 (Structure de jeu temporisé (TGS))**.** Une structure de jeu temporisé (*Timed Game Structure* ou TGS) est un 7-uplet $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ dans lequel :

- L est un ensemble fini de *localités* ;
- $\ell_0 \in L$ est la localité *initiale* ;
- Σ_1 et Σ_2 sont deux *alphabets* d'actions appartenant respectivement au joueur 1 et au joueur 2 ;
- X est un ensemble fini d'*horloges* à valeurs réelles positives ou nulles ;
- $E \subseteq L \times \Sigma \times \mathcal{C}(X) \times 2^X \times L$ est un ensemble fini de *transitions*. Une transition $e = (l,a,g,R,l')$ relie la localité l à la localité l' par l'action a soumise à la *garde* g et mettant à zéro les horloges contenues dans R ;
- $\mathsf{Inv} : L \to \mathcal{B}(X)$ associe un *invariant* à chaque localité.

Exemple 1. La figure 3.1 présente un exemple de TGS. Les contraintes encadrées sont des invariants, les autres des gardes. Une garde ou un invariant non précisé vaut true. La remises à zéro de l'horloge x, par exemple, est écrite explicitement $x := 0$. Le style de dessin des transitions indique leur appartenance : le style plein correspond au joueur 1 et le style interrompu, au joueur 2.

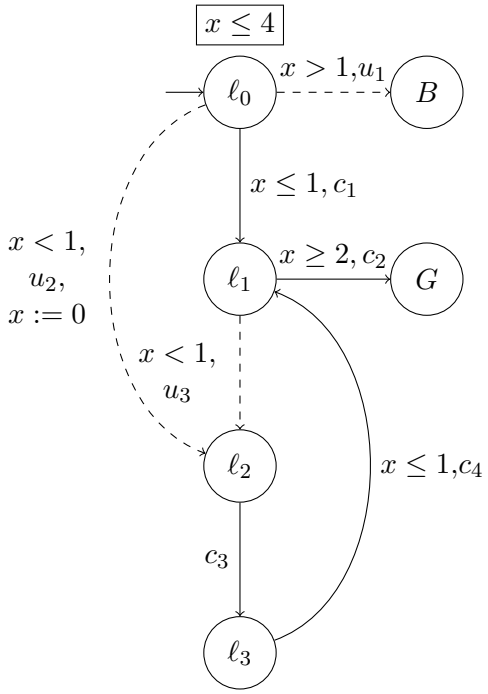


FIGURE 3.1 – Une structure de jeu temporisé

Le comportement d'une structure de jeu temporisé s'exprime naturellement sous la forme

d'un système de transitions temporisé. C'est en fait sur ce système de transitions, généralement infini, que le jeu est joué.

Définition 2 (Sémantique d'une TGS)**.** La sémantique (opérationnelle) d'une TGS $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ est donnée sous la forme d'un système de transitions temporisé $\mathcal{S}_\mathcal{G} = (Q, q_0, \Sigma_1 \cup \Sigma_2 \cup \mathbb{R}_{\geq 0}, \rightarrow)$ où :

- $Q = \{(l,v) | l \in L, v \in \mathbb{R}^X_{\geq 0} \text{ et } v \models \mathsf{Inv}(l)\}$;
- $q_0 = (\ell_0, \mathbf{0}_X)$;
- la relation de transition $\rightarrow \subseteq Q \times \Sigma_1 \cup \Sigma_2 \times Q$ se décompose en :
  - les transitions *discrètes* : $((l_1,v_1),a,(l_2,v_2)) \in \rightarrow$ ssi :
    - $a \in \Sigma_1 \cup \Sigma_2$ et $(l_1,v_1),(l_2,v_2) \in Q$;
    - et il existe une transition $(l_1,a,g,R,l_2) \in E$, telle que $v_1 \models g$ et $v_2 = v_1[R]$.
  - les transitions *temporelles* : $((l_1,v_1),d,(l_2,v_2)) \in \rightarrow$ ssi :
    - $d \in \mathbb{R}_{\geq 0}$ et $(l_1,v_1),(l_2,v_2) \in Q$;
    - $v_2 = v_1 + t$ et pour tout $d' \in \mathbb{R}_{\geq 0}$ t.q. $0 \leq d' \leq d$, $(l_1,v_1 + d') \in Q$.

Nous notons $q \xrightarrow{e} q'$ pour $(q,e,q') \in \rightarrow$.

L'ensemble des comportements possibles des joueurs est ainsi défini. Nous appelerons *exécution* une telle séquence de transitions.

Définition 3 (Exécution)**.** Une *exécution* ρ de $\mathcal{G}$ est une séquence possiblement infinie $q_1,e_1,q_2,e_2,\ldots,e_{n-1},q_n,e_n,\ldots$ telle que pour tout $i \geq 1$, $q_i \xrightarrow{e_i} q_{i+1}$. Une exécution est également notée $\rho = q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} q_n \xrightarrow{e_n} \cdots$.

Exemple 2. Dans la TGS de la figure 3.1, une séquence d'actions possible est : attente de 0,23 unité de temps, suivi de la transition (dont l'action est) u_2 puis attente de 0,34 unité de temps et transition c_3 puis immédiatement transition c_4, attente de 1,5 unité de temps et transition c_2, ce qui nous donne l'exécution suivante (la deuxième composante est la valeur de x) :

$$(\ell_0;0) \xrightarrow{0{,}23} (\ell_0;0{,}23) \xrightarrow{u_2} (\ell_2;0) \xrightarrow{0{,}57} (\ell_3;0{,}57) \xrightarrow{c_3} (\ell_3;0{,}57) \xrightarrow{c_4} (\ell_1;0{,}57) \xrightarrow{1{,}5} (\ell_1;2{,}07) \xrightarrow{c_2} (G;2{,}07)$$

Nous notons $\mathsf{First}(\rho)$ le premier état de ρ. Une exécution ρ est dite *initiale* si $\mathsf{First}(\rho) = (\ell_0,\mathbf{0}_X)$. Si ρ est finie, $\mathsf{Last}(\rho)$ le dernier état de ρ. $\mathsf{States}(\rho)$ désigne l'ensemble des états apparaissant dans ρ. $\mathsf{Duration}(\rho) = \sum_{i \geq 1, e_i \in \mathbb{R}_{\geq 0}} e_i$ est la *durée* de l'exécution, qui peut être infinie.

Nous écrivons $\rho \xrightarrow{e}$ s'il existe un état q tel que $\rho \xrightarrow{q}$. L'ensemble des exécutions de $\mathcal{G}$ dont le premier état est q est noté $\mathsf{Runs}(\mathcal{G},q)$. Le sous-ensemble des exécutions initiales est noté $\mathsf{Runs}(\mathcal{G})$.

3.2.3 Stratégies, résultats et objectifs

Nous nous intéressons à des jeux temporisés à *observation continue* [58] dans lesquels chaque joueur décide de ses actions, à tout moment, en fonction de ce qu'il a pu observer jusque là de l'exécution du système. Ainsi cette exécution est restreinte par les choix des deux joueurs. Cette situation est formalisée par la notion de *stratégie* [39] : chaque joueur a ainsi une stratégie propre qui lui indique, en fonction de l'historique du jeu, ce qu'il doit faire parmi : « jouer une action particulière qui lui appartient » ou « attendre ». Il en résulte une exécution du système conforme aux stratégies des deux joueurs.

Les jeux temporisés considérés ici sont symétriques et nous ne donnerons donc les définitions que du point de vue du joueur 1. Il est possible d'obtenir les définitions correspondantes pour le joueur 2 en échangeant les indices 1 et 2.

Définition 4 (Stratégie)**.** Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS. Une stratégie f sur $\mathcal{G}$ pour le joueur 1 est une fonction totale de l'ensemble des exécutions initiales finies de $\mathcal{G}$ dans $\Sigma_1 \cup \{\mathsf{delay}, \bot\}$ et telle que, pour toute exécution initiale finie ρ, si $f(\rho) \in \Sigma_1$ alors $\rho \xrightarrow{f(\rho)}$ et si $f(\rho) = \mathsf{delay}$ alors il existe $d \in \mathbb{R}_{>0}$ tel que $\rho \xrightarrow{d}$. Finalement, $f(\rho) = \bot$ ssi $\forall e \in \Sigma_1 \cup \mathbb{R}_{>0}, \rho \not\xrightarrow{e}$.

L'historique du jeu est représenté par l'exécution de la TGS jusque là. Le symbole delay indique que le joueur 1 doit attendre. Si la stratégie n'est pas d'attendre alors elle précise l'action à jouer, sauf si ni action ni attente ne sont possibles, ce que nous indiquons avec le symbole $\bot$.

Définition 5 (Stratégie positionnelle)**.** Une stratégie f est *positionnelle* si pour toutes exécutions initiales ρ et ρ', $\mathsf{Last}(\rho) = \mathsf{Last}(\rho')$ implique $f(\rho) = f(\rho')$.

Pour une stratégie positionnelle l'action à jouer ne dépend pas de l'historique complet du jeu mais uniquement de l'état actuel. Ces stratégies sont également dites *sans mémoire* [39, 63].

Exemple 3. Pour la TGS de la figure 3.1, une stratégie (positionnelle) du joueur 1 est : dans les localités ℓ_0, ℓ_2 et ℓ_3 attendre si $x < 1$. Dans ℓ_1, attendre si $x < 2$. Dans $(\ell_0; 1)$ faire l'action c_1, dans $(\ell_2; 1)$ l'action c_3, dans $(\ell_3; 1)$ l'action c_4 et dans ℓ_2 l'action c_2 si $x \leq 2$. Dans tous les autres états, attendre (mais quoi que fasse le joueur 2, ces états ne seront pas atteints).

Si les stratégies des deux joueurs sont fixées, l'exécution résultante du jeu est en général unique : seul le cas où les deux joueurs proposent une action au même moment résulte en un choix non-déterministe. Cependant, nous allons chercher des stratégies pour le joueur 1 qui lui assurent un gain *quelle que soit* la stratégie du joueur 2. Nous allons donc considérer l'ensemble des exécutions possibles résultant de l'application de la stratégie du joueur 1 et d'une stratégie quelconque du joueur 2. Nous appelons cet ensemble d'exécutions *résultat* (*Outcome*) de (l'application de) la stratégie du joueur 1.

Définition 6 (Résultat d'une stratégie)**.** Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS. Soit f une stratégie sur $\mathcal{G}$ pour le joueur 1. Le résultat de l'application de f à $\mathcal{G}$ à partir de l'état q, noté $\mathsf{Outcome}(\mathcal{G},q,f)$, est le sous-ensemble de $\mathsf{Runs}(\mathcal{G},q)$ défini inductivement par :

- $q \in \mathsf{Outcome}(\mathcal{G},q,f)$;
- si $\rho \in \mathsf{Outcome}(\mathcal{G},q,f)$ est finie et $\rho' = \rho \xrightarrow{e} q'$ alors $\rho' \in \mathsf{Outcome}(\mathcal{G},q,f)$ si $\rho' \in \mathsf{Runs}(\mathcal{G},q)$ et l'une des trois conditions suivantes est vérifiée :
  1. $e \in \Sigma_2$;
  2. $e \in \Sigma_1$ et $e = f(\rho)$;
  3. ou $e \in \mathbb{R}_{>0}$ et pour tout $e' \in \mathbb{R}_{\geq 0}$ t.q. $0 \leq e' < e$, il existe $q'' \in Q$ tel que $\rho \xrightarrow{e'} q''$ et $f(\rho \xrightarrow{e'} q'') = \mathsf{delay}$.
- une exécution infinie ρ appartient à $\mathsf{Outcome}(\mathcal{G},q,f)$ si tous ses préfixes finis appartiennent à $\mathsf{Outcome}(\mathcal{G},q,f)$.

Parmi l'ensemble des exécutions résultat de l'application de la stratégie du joueur 1, nous distinguons ceux qui sont infinis ou qui ne peuvent pas être étendus. Ce sont eux qui détermineront le gain ou non pour le joueur 1.

Définition 7 (Résultat maximal)**.** Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS et q un état de (la sémantique de) $\mathcal{G}$. Soit f une stratégie pour le joueur 1. Une exécution $\rho \in \mathsf{Outcome}(\mathcal{G},q,f)$ est *maximale* si :

- soit ρ a un nombre infini de transitions discrètes ;
- soit ρ a un nombre fini de transitions discrètes et :
  - soit il n'existe aucun $e \in \Sigma_1 \cup \Sigma_2 \cup \mathbb{R}_{>0}$ tel que $\rho \xrightarrow{e}$;
  - soit $\mathsf{Duration}(\rho) = +\infty$.

Nous notons $\mathsf{MaxOut}(\mathcal{G}, q, f)$ l'ensemble des exécutions maximales résultats de f à partir de l'état q.

L'ensemble des exécutions possibles du jeu représente toutes les possibilités laissées aux deux joueurs. Parmi celles-ci, nous distinguons certaines qui constituent la *condition de victoire* pour le joueur 1.

Définition 8 (Condition de victoire)**.** Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS. Une *condition de victoire* W pour le joueur 1 est un sous-ensemble de $\mathsf{Runs}(\mathcal{G},(\ell_0,\mathbf{0}_X))$.

Pour gagner, il s'agit alors pour le joueur 1 de restreindre les exécutions du jeu de telle sorte qu'elles fassent parties des exécutions prescrites par la condition de victoire. Cela nous permet de définir la notion de *jeu temporisé*.

Définition 9 (Jeu temporisé)**.** Un *jeu temporisé* est un couple $(\mathcal{G}, W)$ où $\mathcal{G}$ est une TGS et W une condition de victoire.

Définition 10 (Exécutions, stratégies, états et jeux gagnants)**.** Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS. Soit W une condition de victoire. Une exécution initiale ρ est *gagnante* (pour W) si $\rho \in W$.

Une stratégie du joueur 1 est *gagnante* à partir de l'état q si $\mathsf{MaxOut}(\mathcal{G},q,f) \subseteq W$.

Un état q de $\mathcal{G}$ est *gagnant* s'il existe une stratégie du joueur 1 gagnante à partir de q.

Un jeu temporisé $(\mathcal{G}, W)$ est *gagnant* si l'état initial q_0 de $\mathcal{G}$ est gagnant.

Exemple 4. Il est facile de vérifier que la stratégie donnée dans l'exemple 3 est gagnante pour le joueur 1 pour la condition de victoire « atteindre G », c'est-à-dire qui contient toutes les exécutions passant par un état de G.

Nous notons $\mathcal{W}_1(\mathcal{G},\mathsf{Goal})$ l'ensemble des états gagnants pour le joueur 1 dans le jeu temporisé $(\mathcal{G},\mathsf{Goal})$.

Dans la suite, nous considérerons différents types de jeux temporisés caractérisés par leurs conditions de victoires. Soit Goal un ensemble de « bons » états.

- jeux d'*accessibilité* : $(\mathcal{G},W)$ est un jeu d'accessibilité si W est l'ensemble des exécutions ρ telles que $\mathsf{States}(\rho) \cap \mathsf{Goal} \neq \emptyset$. Un tel jeu est gagnant pour le joueur 1 s'il possède une stratégie qui force l'accessibilité de Goal.
- jeux de *sûreté* : $(\mathcal{G},W)$ est un jeu de sûreté si W est l'ensemble des exécutions ρ telles que $\mathsf{States}(\rho) \subseteq \mathsf{Goal}$. Un tel jeu est gagnant pour le joueur 1 s'il possède une stratégie pour rester dans Goal.

– jeux de *Büchi* : $(\mathcal{G},W)$ est un jeu de Büchi si W est l'ensemble des exécutions avec un *nombre infini d'actions discrètes* ρ telles que l'ensemble $\mathsf{States}(\rho) \cap \mathsf{Goal}$ est infini. Un tel jeu est gagnant pour le joueur 1 s'il possède une stratégie qui force la viste de Goal infiniment souvent.

Dans ces trois cas, lorsque cela n'est pas ambigu, nous écrivons $(\mathcal{G},\mathsf{Goal})$ plutôt que $(\mathcal{G},W)$. Par ailleurs, nous savons que pour ces trois conditions de victoires, s'il existe une stratégie gagnante pour le joueur 1, alors il existe une stratégie gagnante *positionnelle* pour le joueur 1 [58, 5].

3.3 Urgence et invariants

3.3.1 Exécutions maximales et invariants stricts

La notion d'exécution résultat maximale est difficile à définir en présence d'invariants stricts. Considérons par exemple la TGS de la figure 3.2. Globalement le temps ne pourra jamais dépasser 1. Cependant, dans tout état de cette TGS, il est toujours possible d'écouler un peu de temps. Ce ne serait pas le cas si l'invariant était non strict : dans l'état $(\ell_0,x = 1)$, il serait alors impossible de laissser passer plus de temps et nous considérerions que toute exécution terminant dans cet état est maximale.

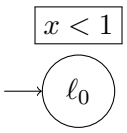


FIGURE 3.2 – Une TGS avec un invariant strict

Afin de régler ce problème, remarquons qu'une TGS possède un nombre fini de transitions et donc de gardes. Et donc, si l'on est « suffisamment proche » de la borne d'un invariant strict, l'ensemble des gardes vraies ne peut plus changer par un écoulement du temps. Plus formellement :

Définition 11 (d-proximité d'un invariant strict)**.** Un état q d'une TGS $\mathcal{G}$ est d-proche d'un invariant strict si $q \overset{d}{\not\to}$ et $\forall d' < d, q \overset{d'}{\to}$. Une exécution finie ρ est d-proche d'un invariant strict si $\mathsf{Last}(\rho)$ l'est.

Lemme 1. *Soit $\mathcal{G}$ une TGS. Il existe $\epsilon > 0$ tel que pour tout état q de $\mathcal{G}$ qui est ϵ-proche d'un invariant strict, pour tout $d < \epsilon$, si $q \overset{d}{\to} q_d$, alors $\forall a \in \Sigma_1 \cup \Sigma_2, q \overset{a}{\to}$ ssi $q_d \overset{a}{\to}$.*

Pour prouver ce résultat, il suffit de remarquer que le nombre de gardes est fini et qu'une fois qu'une garde devient fausse par écoulement du temps elle ne redeviendra plus jamais vraie avant qu'il y ait une transition discrète.

Avant de pouvoir redéfinir les exécutions maximales, il nous faut encore restreindre les conditions de victoires afin qu'elles soient « robustes » vis-à-vis de petits écoulements du temps près des invariants stricts.

Définition 12 (Condition de victoire compatible avec les invariants stricts)**.** Une condition de victoire W sur une TGS $\mathcal{G}$ est *compatible* avec les invariants stricts de $\mathcal{G}$ s'il existe $\epsilon' > 0$ tel que pour toute exécution ρ qui est ϵ'-proche d'un invariant strict de $\mathcal{G}$, $\rho \in W$ ssi toutes les exécutions de $\mathcal{G}$ obtenues par écoulement du temps à partir de ρ appartiennent à W.

À partir de maintenant, en présence d'invariants stricts, nous considérons des conditions de victoire compatibles avec les invariants stricts, ce qui n'est pas une restriction importante.

Soit $\eta = \min(\epsilon,\epsilon')$. Il est maintenant possible de redéfinir les exécutions résultat maximales :

Définition 13 (Résultat maximal (avec invariants stricts))**.** Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS et q un état de $\mathcal{G}$. Soit f une stratégie pour le joueur 1. Une exécution $\rho \in \mathsf{Outcome}(\mathcal{G},q,f)$ est *maximale* si :

- soit ρ a un nombre infini de transitions discrètes ;
- soit ρ a un nombre fini de transitions discrètes et :
  - soit il n'existe aucun $e \in \Sigma_1 \cup \Sigma_2 \cup \mathbb{R}_{>0}$ tel que $\rho \xrightarrow{e}$;
  - soit $\mathsf{Duration}(\rho) = +\infty$;
  - soit ρ est d-proche d'un invariant strict, avec $d < \eta$, et il n'existe aucun $e \in \Sigma_1 \cup \Sigma_2$ et aucun état q tels que $\rho \xrightarrow{e} q \in \mathsf{Outcome}(\mathcal{G},q,f)$.

Le troisième cas que nous avons ajouté correspond à des exécutions qui ne sont pas des blocages complets car le temps peut continuer à s'écouler, mais puisque $d < \eta$, nous savons que, par simple écoulement du temps, aucune garde ne changera de valeur et l'état « gagnant ou non » de l'exécution ne changera pas non plus.

3.3.2 Urgence et invariants stricts

Quand un joueur ne veut pas jouer l'une de ses actions disponibles, il a en général la possibilité d'attendre. Ce n'est cependant pas le cas si le jeu se trouve dans une localité pourvue d'un invariant non-trivialement vrai. Considérons la TGS de la figure 3.3. Pour toute exécution finissant dans l'état $(\ell_0,x = 1)$, la définition 4 interdit delay et impose de choisir une action discrète. Si la condition de victoire pour le joueur 1 est « atteindre G », le jeu est alors gagnant pour le joueur 1 car le joueur 2 est forcé de jouer u quand x vaut 1.

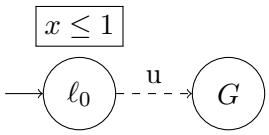


FIGURE 3.3 – Une TGS avec action forcée

Cependant, si dans le même exemple l'invariant est *strict*, la définition 4 n'impose plus cette contrainte car delay est possible dans tous les états dont la localité est ℓ_0.

Pour prendre en compte la contrainte *globale* de l'invariant qui dit que « il faut quitter ℓ_0 avant que x ne vaille 1 », il faut modifier légèrement la définition 4.

Définition 14 (Stratégie (avec invariants stricts))**.** Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS. Une stratégie f sur $\mathcal{G}$ pour le joueur 1 est une fonction totale de l'ensemble des exécutions initiales finies de $\mathcal{G}$ dans $\Sigma_1 \cup \{\mathsf{delay},\bot\}$ et telle que, pour toute exécution initiale finie ρ

– si $f(\rho) \in \Sigma_1$ alors $\rho \xrightarrow{f(\rho)}$
– si $f(\rho) = \mathsf{delay}$ alors il existe $d \in \mathbb{R}_{>0}$ tel que $\rho \xrightarrow{d}$. Et si ρ est d'-proche d'un invariant strict, avec $d' < \eta$, et s'il existe $a \in \Sigma_1$, tel que $\rho \xrightarrow{a}$, alors il existe $d'' < d'$ et un état q tel que $f(\rho \xrightarrow{d''} q) \neq \mathsf{delay}$.

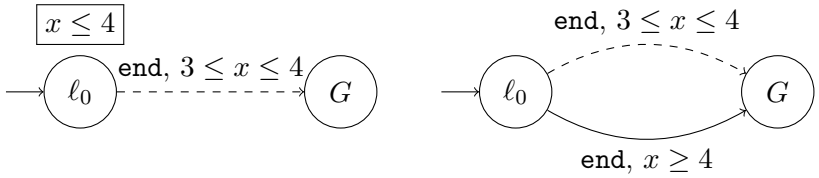
Finalement, $f(\rho) = \bot$ ssi $\forall e \in \Sigma_1 \cup \mathbb{R}_{>0}, \rho \not\xrightarrow{e}$.

La restriction supplémentaire apportée par cette nouvelle définition permet ainsi d'empêcher d'attendre indéfiniment lorsqu'on est proche d'un invariant strict.

3.3.3 Équivalence avec les jeux sans invariants

Les invariants et le comportement mis en évidence par l'exemple de la figure 3.3 sont très utiles dans la modélisation en pratique : supposons que l'on souhaite modéliser un tapis roulant à la vitesse incertaine.

Considérons la figure 3.4a : nous modélisons l'arrivée d'un produit au bout du tapis roulant par une transition `end`.



(a) (b)

FIGURE 3.4 – Deux TGS modélisant un tapis roulant

La garde de cette transition est un intervalle pour exprimer l'incertitude sur la vitesse du tapis. De toute évidence la transition ne peut pas être contrôlable (appartenir au joueur 1) car cela reviendrait à considérer qu'on en choisit la vitesse. Donc elle est incontrôlable (appartient au joueur 2).

Sans l'invariant sur ℓ_0, le joueur 2 peut très bien choisir de ne jamais jouer `end` avec une stratégie d'attente permanente.

Les invariants posent cependant un problème théorique important puisque, comme le montrent les exemples précédents, l'intuition provenant des jeux non temporisés et qui est à la base des algorithmes de calcul des états gagnants, qui veut qu'un état est gagnant pour le joueur 1 s'il existe une de ses transitions qui mène à un état gagnant et si toutes les transitions du joueur 2 mènent également à un état gagnant, n'est plus vérifiée.

Dans l'exemple de la figure 3.4, une alternative à l'invariant est présentée sur la figure 3.4b. Il s'agit ici d'ajouter une transition du joueur 1 à la date 4 qui permette de forcer l'occurrence de la transition `end`. Il serait possible d'aller encore plus loin et de ne pas ajouter cette transition mais de changer l'objectif du joueur 1 en « atteindre G ou ℓ_0 avec $x > 1$ ». L'idée est dans les deux cas de « punir » le joueur 2 s'il ne respecte pas la contrainte dictée par l'invariant, en faisant gagner le joueur 1.

Nous proposons maintenant une généralisation de cette transformation qui nous permet de ne considérer que des jeux sans invariants. Une telle transformation a également l'avantage de supprimer les invariants stricts qui compliquent les définitions.

Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS et W une condition de victoire sur $\mathcal{G}$. Soit $\mathcal{G}' = (L, \ell_0, \Sigma_1, \Sigma_2, X, E', \mathsf{Inv}')$ une TGS construite à partir de $\mathcal{G}$ de la façon suivante :

- pour chaque localité l, nous renforçons (en conjonction) la garde de chaque transition sortante par $\mathsf{Inv}(l)$;
- pour chaque localité l, nous renforçons (en conjonction) la garde de chaque transition entrante par la projection de $\mathsf{Inv}(l)$ sur les horloges non remises à zéro ;
- pour toute localité l, $\mathsf{Inv}'(l) = \mathsf{true}$.

Nous noterons $\rightarrow_{\mathcal{G}}$ la relation de transition de la sémantique de $\mathcal{G}$ et $\rightarrow_{\mathcal{G}'}$ celle de la sémantique de $\mathcal{G}'$.

Remarquons que $\mathsf{Runs}(\mathcal{G}') = \mathsf{Runs}(\mathcal{G}) \cup R$, où R disjoint de $\mathsf{Runs}(\mathcal{G})$ est l'ensemble des exécutions qui violent les invariants de $\mathcal{G}$: pour tout $\rho' \in R$, il existe une exécution $\rho \in \mathsf{Runs}(\mathcal{G})$, $d \in \mathbb{R}_{>0}$ et un état q de $\mathcal{G}'$ tels que $\rho' = \rho \xrightarrow{d} q$.

Nous définissons maintenant une nouvelle condition de victoire W' par $W' = W^{\nearrow} \cup R'$ où :

- $W^{\nearrow}$ est obtenu depuis W en étendant par un écoulement infini du temps toutes les exécutions de W qui ont un nombre fini d'actions discrètes et une durée finie, c.-à-d. celles pour lesquelles le temps était bloqué par un invariant dans $\mathcal{G}$;
- R' est le sous-ensemble de R contenant les exécutions dans lesquelles le joueur 2 était le seul qui aurait pu empêcher la violation d'un invariant de $\mathcal{G}$: $\rho' \in R'$ s'il existe $\rho \in \mathsf{Runs}(\mathcal{G})$, $d \in \mathbb{R}_{>0}$ et un état q de $\mathcal{G}'$ tels que :
  - $\rho' = \rho \xrightarrow{d}_{\mathcal{G}'} q$;
  - $\forall d' \in \mathbb{R}_{>0}, \rho \xrightarrow{d'}$ ou ρ est η-proche d'un invariant strict ;
  - $\exists a \in \Sigma_2$ tel que $\rho \xrightarrow{a}_{\mathcal{G}}$ et $\forall b \in \Sigma_1, \rho \not\xrightarrow{b}_{\mathcal{G}'}$.

L'ensemble R' permet, comme évoqué précédemment, de « punir » le joueur 2 s'il ne respecte pas les invariants.

Proposition 1. *Si f est une stratégie gagnante pour $(\mathcal{G}',W')$ alors sa restriction $f_{\mathcal{G}}$ à $\mathsf{Runs}(\mathcal{G})$ est une stratégie gagnante pour $(\mathcal{G},W)$.*

Proposition 2. *Soit f une stratégie gagnante pour $(\mathcal{G},W)$. Soit f' la stratégie définie par $f'(\rho) = \mathsf{delay}$ si $f(\rho) = \bot$ ou $\rho \notin \mathsf{Runs}(\mathcal{G})$ et $f'(\rho) = f(\rho)$ sinon. Alors f' est une stratégie gagnante pour $(\mathcal{G}',W')$.*

Puisqu'il s'agit principalement d'ajouter des exécutions des états gagnants correspondant à la violation des invariants par le joueur 2, il est facile de voir que la transformation proposée « préserve » les jeux d'accessibilités au sens ou si $(\mathcal{G},W)$ est un jeu d'accessibilité alors $(\mathcal{G}',W')$ tel que défini ci-dessus l'est aussi. Il en est de même pour les jeux de sûreté. Pour les jeux de Büchi, c'est légèrement moins immédiat et il faut considérer que le joueur 1 dispose de transitions bouclant sur toute localité avec une garde égale à la négation de l'invariant pour formellement retrouver un jeu de Büchi après la transformation.

3.4 Jeux d'accessibilité temporisés

Nous nous concentrons maintenant sur le calcul des états gagnants pour les jeux d'accessibilité temporisés. Ce calcul se base sur la notions de *prédécesseurs contrôlables* [58, 39].

3.4.1 Prédecesseurs contrôlables

Un état est gagnant pour le joueur 1 si celui-ci peut atteindre un état gagnant en laissant écouler du temps puis éventuellement en prenant l'une de ses transitions, et si toutes les transitions du joueur 2 possibles sur ce chemin mènent également à un état gagnant pour le joueur 1.

L'idée est alors de partir de Goal et de calculer « en arrière » les prédecesseurs des états gagnants par une action du joueur 1 qui ne sont pas prédecesseurs d'états non gagnants par une action du joueur 2. Nous définissons donc les prédécesseurs d'un ensemble d'états :

Définition 15 (Prédécesseur discret d'un ensemble d'états)**.** Soit $V \subset Q$ un ensemble d'états de $\mathcal{G}$ et $e = (l,a,g,R,l')$ une transition d'une TGS. Les prédécesseurs discrets de V par e sont $\mathsf{Pred}(V,e) = \{(l,v) \in Q | v[R] \in V\}$.

Pour $i \in \{1,2\}$, nous notons $\mathsf{Pred}_i(S')$ l'union de tous les prédécesseurs de S' par une transition du joueur i.

L'opérateur Pred permet de trouver les états qui par franchissement (instantané) d'une transition amènent dans S'. Il nous faut maintenant prendre en compte l'écoulement du temps. [58] définit la notion de prédecesseurs temporels d'un ensemble d'états U sûrs (pour le joueur 1) vis-à-vis d'un ensemble d'états V, notée $\mathsf{Pred}_t(S,T)$. Intuitivement, ce sont les états à partir desquels il est possible d'atteindre S par écoulement du temps sans passer par T.

Définition 16. Les *prédécesseurs temporels* de U *en évitant* V sont :

$$\mathsf{Pred}_t(U,V) = \{q \in Q | \exists d \in \mathbb{R}_{\geq 0} \text{ t.q. } q \xrightarrow{d} q', q' \in U \text{ et } \mathsf{Post}_{[0,d]}(q) \subseteq \overline{V}\}$$

avec $\mathsf{Post}_{[0,d]}(q) = \{q' \in Q | \exists d' \in [0,d] \text{ t.q. } q \xrightarrow{d'} q'\}$.

Nous en arrivons à la notion de prédecesseurs *contrôlables* dont l'intuition a été donnée au début de cette section :

Définition 17. L'opérateur *prédecesseurs contrôlables* est la fonction $\pi : 2^Q \to 2^Q$ telle que :

$$\pi(V) = \mathsf{Pred}_t(V \cup \mathsf{Pred}_1(V), \mathsf{Pred}_2(\overline{V}))$$

Soit $(\mathcal{G},\mathsf{Goal})$ un jeu temporisé d'accessibilité. Le calcul itératif $W^0 = \emptyset$ et $W^{n+1} = \mathsf{Goal} \cup \pi(W^n)$ converge après un nombre fini d'étapes et le plus petit point fixe obtenu, W^* est exactement l'ensemble des états gagnants de $(\mathcal{G},\mathsf{Goal})$ [58]. Ainsi, il existe une stratégie gagnante (positionnelle) ssi $(\ell_0,\mathbf{0}_X) \in W^*$. L'extraction des stratégies peut se faire à partir de W^*.

Remarque 1. Il est plus usuel de définir π selon la formule $\pi(X) = \mathsf{Pred}_t(\mathsf{Pred}_1(X),\mathsf{Pred}_2(X))$. Cela impose au joueur 1 de terminer par une action discrète pour gagner. Ici nous voulons que l'on puisse également gagner par simple écoulement du temps. Ainsi le joueur 1 peut gagner sur la TGS présentée sur la figure 3.5 pour le jeu d'accessibilité défini par $\mathsf{Goal} = (\ell_0, x \geq 2)$.

Cela implique que les prédécesseurs temporels des états déjà gagnants, évitant les prédécesseurs par des transitions du joueur 2 d'états non (encore) gagnants, doivent aussi être gagnants, ce qui est reflété par la formule $\pi(X) = \mathsf{Pred}_t(X \cup \mathsf{Pred}_1(X),\mathsf{Pred}_2(X))$.

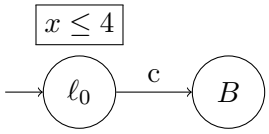


FIGURE 3.5 – Une TGS avec action forcée

3.4.2 Graphe de simulation

Le point fixe que nous venons de présenter a un défaut principal : il calcule des états qui ne sont pas accessibles dans la TGS. Pour un modèle d'automates temporisés enrichi comme celui d'UPPAAL [56], qui permet notamment l'utilisation de variables discrètes en plus des localités ce problème est amplifié : si la variable entière i subit une affectation du type $i := 2$ lors du franchissement d'une transition, l'ensemble des prédécesseurs discrets possibles de cette transition est potentiellement énorme car il correspond à toutes les valeurs possibles de i.

Une solution est de calculer l'ensemble des états accessibles de la TGS et d'en faire l'intersection avec W^n à chaque itération.

L'ensemble infini des états accessibles d'un automate temporisé, et donc d'une TGS, peut être partitionné en un nombre fini d'*états symboliques* regroupant un nombre potentiellement infini de valuations des horloges, tout en préservant certaines propriétés.

Ici nous souhaitons préserver l'accessibilité des états. L'automate des régions [3] réalise une telle partition mais nous nous basons ici sur le graphe de simulation [55] qui, bien que potentiellement beaucoup plus gros en théorie, fournit généralement un graphe bien plus petit en pratique.

Dans le graphe de simulation, les états qui peuvent être obtenus par la même séquence d'actions discrètes sont regroupés. Ils partagent donc la même localité, ce qui conduit à la notion d'*état symbolique*.

Définition 18 (État symbolique)**.** Soit $\mathcal{G} = (L, \ell_0, \Sigma_1, \Sigma_2, X, E, \mathsf{Inv})$ une TGS. Un *état symbolique* de $\mathcal{G}$ est un ensemble d'états qui partagent la même localité. C'est donc un couple (l,V) dans lequel l est une localité de $\mathcal{G}$ et $V \subseteq \mathbb{R}^X_{\geq 0}$ un ensemble de valuations.

Trois opérations spécifiques sont définies sur les ensembles de valuations :
- futur : $V^{\nearrow} = \{v + d | v \in V, d \in \mathbb{R}_{\geq 0}\}$;
- passé : $V^{\swarrow} = \{v' \in \mathbb{R}^X_{\geq 0} | v' + d = v, v \in V, d \in \mathbb{R}_{\geq 0}\}$;
- mise à zéro d'un sous-ensemble d'horloges $R \subseteq X$: $V[R] = \{v[R] | v \in V\}$.

Il est possible de calculer *symboliquement* l'état symbolique successeur par une transition de la TGS d'un autre état symbolique :

Définition 19 (Successeur discret d'un état symbolique)**.** Soit $S = (l,Z)$ un état symbolique et $e = (l,a,g,R,l')$ une transition de $\mathcal{G}$. Le successeur discret de S par e, noté $\mathsf{Succ}(S,e)$, est l'état symbolique $(l',(Z \cap [\![g]\!])[R])$.

Le successeur discret contient les états pouvant être atteints immédiatement en franchissant la transition depuis S. Il faut encore ajouter les états que l'on peut obtenir à partir du successeur discret par écoulement du temps.

Définition 20 (Successeur symbolique d'un état symbolique)**.** Soit $S = (l,Z)$ un état symbolique et $e = (l,a,g,R,l')$ une transition de $\mathcal{G}$. Le successeur symbolique de S par e, noté $\mathsf{Next}(S,e)$, est l'état symbolique $(l',\mathsf{Succ}(S,e)^{\nearrow} \cap [\![\mathsf{Inv}(l')]\!])$.

Nous notons également $S \xrightarrow{e} S'$ si $S' = \mathsf{Next}(S,e)$. Nous calculons alors le graphe de simulation à partir de l'état symbolique initial $S_0 = (\ell_0, \{\mathbf{0}_X\}^{\nearrow})$ en calculant avec Next tous les états symboliques accessibles.

Si toutes les horloges sont bornées, il y a un nombre fini d'états symboliques, sinon il faut légèrement modifier Next. On pourra alors se référer, par exemple, à [13] pour une explication plus complète.

Dans le cas des automates temporisés, nous pouvons montrer que les ensembles de valuations des états symboliques ont une forme bien particulière appelée *zone*.

Définition 21 (Zone)**.** Une *zone* est un ensemble de valuations $Z \subseteq \mathbb{R}^X_{\geq 0}$ pour lequel il existe une contrainte $g \in \mathcal{Z}(X)$ telle que $Z = [\![g]\!]$.

Cette forme particulière peut être représentée par des matrices de différences bornées (*Difference Bound Matrix*, ou DBM) [12, 40]. Nous disposons sur les DBM d'algorithmes efficaces (quadratiques en le nombre d'horloges) pour le calcul des opérations symboliques impliquées dans la génération des états symboliques.

Exemple 5. La figure 3.6 présente le graphe de simulation de la TGS de la figure 3.1. Remarquons que ce graphe considéré comme un jeu non temporisé est perdant pour le joueur 1 et l'objectif « atteindre G », à cause de la transition u_1. Nous avons pourtant vu avec l'exemple 4 qu'il existe bien une stratégie gagnante pour le jeu temporisé correspondant.

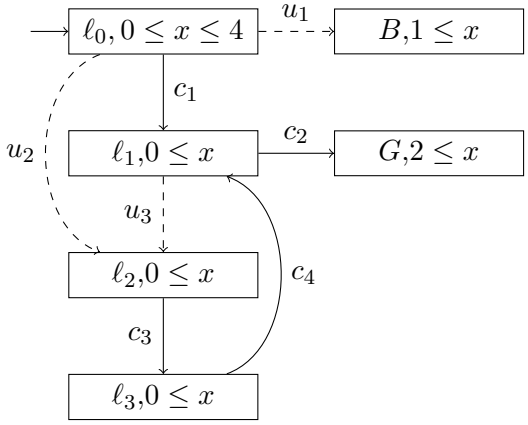


FIGURE 3.6 – Graphe de simulation de la TGS de figure 3.1

L'exemple 5 montre bien que le graphe de simulation n'est pas en lui-même une abstraction assez fine pour décider si un jeu est gagnant ou pas. En effet, les états symboliques ne sont pas homogènes vis-à-vis de l'ensemble des états gagnants : dans la suite nous verrons comment calculer le sous-ensemble $\mathrm{Win}[S]$ des états gagnants d'un état symbolique S.

3.4.3 Calcul du point fixe à la volée

L'intersection avec l'ensemble des états accessibles permet d'envisager l'utilisation en pratique du point fixe de la section 3.4.1, notamment avec le formalisme étendu utilisé par UPPAAL.

Cependant, l'un des intérêts fondamentaux des algorithmes d'UPPAAL, et qui a fait ses preuves en pratique est, lors de la vérification de propriétés de logique temporelle, la possibilités, dans certains cas, de terminer avant d'avoir calculé tout l'espace d'états. Le cas le plus simple d'une telle terminaison anticipée est la vérification de l'accessibilité d'un état q. Dès qu'un état symbolique contenant q est généré, l'algorithme peut s'arrêter. Bien sûr, si q n'est pas accessible, il faudra tout de même calculer tout l'espace d'états pour s'en rendre compte. Cependant, en pratique, cette possiblité de terminaison anticipée peut permettre des gains de temps importants, voire de vérifier des propriétés qui seraient trop coûteuses en mémoire ou en temps de calcul s'il fallait calculer l'espace d'états en entier.

La méthode de calcul des états gagnants proposée à la section précédente peut également être interrompue de façon anticipée, dès que l'état initial de la TGS fait partie des états gagnants. Elle implique cependant de calculer préalablement tout l'espace d'états, ce qui est une contrainte importante en termes de temps de calcul et d'espace mémoire requis.

Nous proposons donc maintenant un algorithme qui permet de réaliser le calcul des états accessibles et la propagation en arrière des états gagnants de façon entremêlée et relativement désynchronisée, ce qui permettra dans certains cas de prouver que l'état initial est gagnant sans calculer tout l'espace d'états.

Algorithme 1 L'algorithme SOTFBLF (**S**ymbolic **O**n-**T**he-**F**ly Algorithm for **B**ackwards **L**east **F**ixpoint Computation)

1: $S_0 \leftarrow \{(\ell_0, \mathbf{0}_X)\}^{\nearrow}$
2: Passed $\leftarrow \{S_0\}$
3: Waiting $\leftarrow \{(S_0, e, S') | S' = \mathsf{Next}(S_0, e)\}$
4: Win$[S_0] \leftarrow S_0 \cap F(\emptyset)$
5: Depend$[S_0] \leftarrow \emptyset$
6:
7: **while** Waiting $\neq \emptyset$ and $(\ell_0, \mathbf{0}_X) \notin$ Win$[S_0]$ **do**
8: $\quad \sigma = (S, e, S') \leftarrow pop$(Waiting)
9: $\quad$ **if** $S' \notin$ Passed **then**
10: $\quad\quad$ Passed $\leftarrow$ Passed $\cup \{S'\}$
11: $\quad\quad$ Depend$[S'] \leftarrow \{\sigma\}$
12: $\quad\quad$ Win$[S'] \leftarrow S' \cap F(\emptyset)$
13: $\quad\quad$ Waiting $\leftarrow$ Waiting $\cup \{(S', e', S'') | S'' = \mathsf{Next}(S', e')\}$
14: $\quad\quad$ **if** Win$[S'] \neq \emptyset$ **then**
15: $\quad\quad\quad$ Waiting $\leftarrow$ Waiting $\cup \{\sigma\}$
16: $\quad\quad$ **end if**
17: $\quad$ **else**
18: $\quad\quad$ Win$^* \leftarrow S \cap F($Win$[S] \cup \bigcup_{e' \in E}$ Win$[\mathsf{Next}(S, e')])$
19: $\quad\quad$ **if** Win$[S] \subsetneq$ Win* **then**
20: $\quad\quad\quad$ Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$
21: $\quad\quad\quad$ Win$[S] \leftarrow$ Win*
22: $\quad\quad$ **end if**
23: $\quad\quad$ Depend$[S'] \leftarrow$ Depend$[S'] \cup \{\sigma\}$
24: $\quad$ **end if**
25: **end while**

Soit $F : 2^Q \to 2^Q$ une fonction sur les ensembles d'états vérifiant les trois propriétés

suivantes :

F1 F est croissante ;

F2 si un état q appartient à $F(V)$ pour $V \subseteq Q$ alors il existe $U \subseteq \{q\}^{\nearrow} \cup \bigcup_{e \in E} \mathsf{Succ}(\{q\}^{\nearrow}, e)$ tel que $U \subseteq V$ et $q \in F(U)$.

F3 si V est une union finie de régions alors $F(V)$ aussi.

La propriété **F2** indique que l'appartenance de q à F ne doit dépendre que des états obtenus à partir de q par écoulement du temps *éventuellement* suivi d'une seule transition discrète. Il est facile de voir que l'opérateur π, par exemple, vérifie ces conditions.

Soit $\mu X.F(X)$ le plus petit point fixe de la fonction F. Pour calculer $Q \cap \mu X.F(X)$ nous proposons l'algorithme 1.

Il peut être vu comme une extension symbolique de l'algorithme de Liu et Smolka présenté dans [57]. Il se décompose en deux phases : une exploration en avant qui construit le graphe de simulation (lignes 10 à 16) et une propagation en arrière des sous-ensembles des états de ce graphe appartenant au point fixe rencontrés (lignes 18 à 23). Les deux phases ne sont que très peu synchronisées ce qui permet une terminaison anticipée quand $(\ell_0, \mathbf{0}_X) \in \mathrm{Win}[S_0]$ sans avoir construit l'intégralité du graphe de simulation ni même forcément propagé les états du point fixe à tous les états symboliques déjà calculés.

Nous utilisons plusieurs ensembles dans cet algorithme : Passed est l'ensemble des états symboliques déjà calculés.

Waiting est la liste d'attente des états symboliques à examiner, à la fois en avant et en arrière.

Win donne, pour chaque état symbolique, le sous-ensemble de ses états qui appartiennent au point fixe. Le calcul local du point fixe est fait ligne 18 et initialisé aux lignes 4 et 12 pour chaque nouvel état symbolique calculé. Nous supposons que pour tout état symbolique S non encore calculé, $\mathrm{Win}[S] = \emptyset$.

Depend donne pour chaque état symbolique S l'ensemble de ses prédécesseurs dans le graphe de simulation. Il faut recalculer Win pour chacun d'entre eux quand $\mathrm{Win}[S]$ grossit (lignes 19 et 20).

L'ordre d'exploration - propagation en arrière, imposé par la fonction *pop* est indifférent et de nombreuses stratégies sont possibles : propager immédiatement tout nouvel état du point fixe trouvé, attendre afin de minimiser le nombre de propagations en arrière, etc.

Nous prouvons maintenant la terminaison et la correction de l'algorithme 1.

Correction de l'algorithme

La correction de l'algorithme SOTFBLF est donnée par le théorème suivant :

Théorème 1. *Si F vérifie les propriétés* **F1** *et* **F2***, alors à la terminaison de l'exécution de l'algorithme SOTFBLF sur une TGS, les deux propriétés suivantes sont vraies :*

1. *si $q \in \mathrm{Win}[S]$ pour un $S \in$ Passed alors $q \in \mu X.F(X)$;*
2. *si $\mathrm{Waiting} = \emptyset$ et $q \in S \setminus \mathrm{Win}[S]$ pour un $S \in$ Passed alors $q \notin \mu X.F(X)$.*

Terminaison de l'agorithme

Le théorème suivant assure la terminaison de l'algorithme SOTFBLF :

Théorème 2. *Si F vérifie les propriétés* **F1** *et* **F3** *alors la terminaison de l'algorithme SOTFBLF est garantie.*

L'algorithme SOTFTR pour les jeux d'accessibilité temporisés

Soit F_r la fonction telle que $F_r(X) = \mathsf{Goal} \cup \pi(X)$. Il est facile de voir que F_r vérifie bien les propriétés **F1** et **F2** imposées à F dans l'agorithme SOTFBLF. Si Goal est un ensemble fini de régions alors, avec les théorèmes 3 et 4 présentés ci-après, nous pouvons conclure que F_r vérifie également **F3**.

Nous appelons SOTFTR (*Symbolic On-The-Fly Algorithm for Timed Reachability games*) l'agorithme SOTFBLF instancié avec la fonction F_r. D'après tout ce qui précède, cet algorithme calcule donc les états gagnants d'un jeu d'accessibilité temporisé, avec une possibilité de terminaison anticipée.

3.4.4 Utilisation des zones pour l'opérateur Pred_t

L'autre avantage important de l'algorithme SOTFBLF est qu'il fonctionne en manipulant uniquement des zones, pour lesquelles des algorithmes efficaces de manipulation existent.

Il est facile de montrer que l'intersection de deux zones est une zone. La différence de deux zones, par contre, est une union finie de zones.

Nous avons vu que Next fournit des zones. Par ailleurs, si $S = (l,Z)$ est un état symbolique et $e = (l,a,g,R,l')$ une transition de la TGS, alors nous pouvons très facilement calculer $\mathsf{Pred}(\mathsf{Next}(S,e),e)$: c'est l'état symbolique $(l,Z \cap [\![g]\!])$. L'ensemble de ses valuations est donc bien une zone.

Nous montrons maintenant comment Pred_t fournit une union finie de zones. Nous montrons donc maintenant comment manipuler ces unions (théorème 3) et comment définir Pred_t avec les opérations classiques sur les zones (théorème 4).

Théorème 3 ([C17]). *La loi de distribution suivante est vraie :*

$$\mathsf{Pred}_t(\bigcup_i G_i, \bigcup_j B_j) = \bigcup_i \bigcap_j \mathsf{Pred}_t(G_i,B_j)$$

Théorème 4 ([C17]). *Si B est convexe alors l'opérateur Pred_t peut s'écrire :*

$$\mathsf{Pred}_t(G,B) = (G^{\swarrow} \setminus B^{\swarrow}) \cup ((G \cap B^{\swarrow}) \setminus B)^{\swarrow}$$

3.5 Autres jeux temporisés

3.5.1 Jeux d'accessibilité en temps minimal

Dans un jeu d'accessibilité en temps minimal, nous cherchons la stratégie qui assure, non seulement d'atteindre Goal, mais encore de l'atteindre le plus rapidement possible : si t^* est le temps minimal, alors le joueur 1 a une stratégie pour atteindre Goal en au plus t^* unités de temps, quoi que fasse le joueur 2, mais de plus, il n'existe pas de stratégie du joueur qui puisse garantir cela pour $t < t^*$.

L'algorithme SOTFTF peut très facilement être modifié pour calculer t^* : il suffit d'ajouter une horloge z et un invariant $z \leq B$ pour toutes les localités à la TGS. Dans l'algorithme,

l'état symbolique initial est alors (ℓ_0,Z_0) où Z_0 est obtenue à partir de $\mathbf{0}_X$ en relâchant toutes les contraintes sur z. Par ailleurs, il faut désactiver la terminaison anticipée de l'algorithme.

Nous calculons alors l'ensemble des états gagnants et vérifions que $(\ell_0, \mathbf{0}_X)$ est bien gagnant. Si ce n'est pas le cas nous pouvons réessayer avec $B' > B$, sinon nous savons qu'il y a une stratégie pour atteindre Goal en moins de B unités de temps et, de plus, les valeurs possibles de z nous donnent le temps minimal. En effet soit $I = \mathrm{Win}[Z_0] \cap \{(\ell_0, \mathbf{0}_{X\setminus\{z\}})\}$. I est un intervalle tel que $0 \in I$ et si b est la borne maximale de I alors $b \leq B$.

Par définition de l'ensemble maximal des états gagnants calculés par l'algorithme, nous savons qu'en partant de $(\ell_0, \mathbf{0}_{X\setminus\{z\}})$ avec $z \in I$ il est possible d'atteindre Goal en moins de B unités de temps et si $z \notin I$ alors c'est impossible. Le temps minimal est donc $t^* = B - b$ et si I est ouvert en b, nous savons qu'aucune stratégie ne peut réaliser ce minimum mais qu'on peut s'en approcher arbitrairement près.

3.5.2 Jeux de sûreté

Un jeu de sûreté $(\mathcal{G}, \mathsf{Goal})$ consiste, pour le joueur 1, à trouver une stratégie pour rester dans Goal. Il est possible de reformuler ce problème en donnant, de façon duale, un ensemble Bad d'états à éviter : $\mathsf{Bad} = \mathsf{States}(\mathsf{Runs}(\mathcal{G})) \setminus \mathsf{Goal}$. Il s'agit alors, pour le joueur, de trouver une stratégie pour éviter Bad quelles que soient les actions du joueur 2.

Dans [58], les auteurs montrent que les états gagnants de $(\mathcal{G},\mathsf{Goal})$ peuvent être calculés comme le plus grand point fixe $\nu X.\mathsf{Goal} \cap \pi(X)$. Nous allons nous ramener à un plus petit point fixe pour pouvoir utiliser l'algorithme SOTFBLF.

Soit Q l'ensemble des états accessibles de la TGS $\mathcal{G}$. Nous calculons $\overline{\mathcal{W}_1(\mathcal{G},\mathsf{Goal})}$, le complémentaire de $\mathcal{W}_1(\mathcal{G},\mathsf{Goal})$ dans Q. Il s'agit des états non gagnants pour le joueur 1 ou encore ceux à partir desquels le joueur 2 à une stratégie pour « gâcher » la partie du joueur 1 sans pour autant avoir forcément lui-même une stratégie gagnante (*Spoiling strategy*) [38].

Nous avons donc :

$$\overline{\mathcal{W}_1(\mathcal{G}, \mathsf{Goal})} = \overline{\nu X.\mathsf{Goal} \cap \pi(X)} = \mu X.\overline{\mathsf{Goal}} \cup \overline{\pi(\overline{X})} = \mu X.\mathsf{Bad} \cup \overline{\pi(\overline{X})}$$

Par ailleurs,

$$\overline{\pi(\overline{X})} = \overline{\mathsf{Pred}_t(\overline{X} \cup \mathsf{Pred}_1(\overline{X}),\mathsf{Pred}_2(X))}$$

Soit encore

$$\overline{\pi(\overline{X})} = \overline{\mathsf{Pred}_t}(\overline{X} \cup \mathsf{Pred}_1(\overline{X}),\mathsf{Pred}_2(X))$$

avec :

$$\overline{\mathsf{Pred}_t}(U,V) = \{q \in Q | \forall d \in \mathbb{R}_{\geq 0}, (q \xrightarrow{d} q' \text{ et } q' \notin U) \text{ ou } \mathsf{Post}_{[0,d]}(q) \cap V \neq \emptyset\}$$

Notons $\overline{\pi}(X) = \overline{Pred_t}(X \cup \mathsf{Pred}_1(X),\mathsf{Pred}_2(\overline{X}))$. Alors il est facile de voir que, tout comme F_r, la fonction $F_s(X) = \mathsf{Bad} \cup \overline{\pi}(\overline{X})$ vérifie les propriétés **F1**, **F2** et **F3**. Il est donc possible utiliser l'agorithme SOTFBLF pour calculer $\mu X.F_s(X)$.

Nous appelons SOTFTS pour *Symbolic On-The-Fly Algorithm for Timed Safety Games* l'algorithme SOTFBLF instancié avec la fonction F_s.

Par ailleurs, nous déduisons aisément des théorèmes 3 et 4 les résultats similaires permettant de calculer $\overline{\mathsf{Pred}_t}$ avec des zones.

3.5.3 Jeux de Büchi

Dans un jeu de Büchi ($\mathcal{G}$,Goal) le joueur 1 doit forcer la visite infiniment souvent des états de Goal. Nous imposons par ailleurs que ce faisant il y ait une infinité d'actions discrètes (pas de séjour infiniment long dans Goal sans action discrète si celui-ci est clos par écoulement du temps).

Par rapport à un jeu d'accessibilité, il faut, pour le joueur 1, non seulement forcer l'accessibilité d'un état Goal, mais aussi, à partir de cet état, forcer l'accessibilité d'un état de Goal et ainsi de suite. Il n'est donc pas étonnant que l'ensemble des états gagnants du jeu ($\mathcal{G}$,Goal) puisse alors être calculé par la résolution successive de plusieurs jeux d'accessibilité. [58] propose le point fixe suivant pour résoudre les jeux de Büchi :

$$\nu Y.\mu X.\pi'(X) \cup (\mathsf{Goal} \cap \pi'(Y))$$

avec $\pi'(X) = \mathsf{Pred}_t(\mathsf{Pred}_1(X),\mathsf{Pred}_2(\overline{X}))$.

Intuitivement, et en simplifiant un peu, le point fixe interne sur X calcule l'ensemble des états qui peuvent forcer Goal et le point fixe externe sur Y enlève au fur et à mesure de Goal les états qui ne peuvent pas eux même forcer Goal en au moins une action discrète.

L'idée est alors d'utiliser l'algorithme SOTFBLF pour calculer le point fixe sur X. Néanmoins, ici il ne faut pas s'arrêter de façon anticipée si l'état initial est dans X, puisque c'est bien l'ensemble complet des états du point fixe qui nous intéressent. En supprimant la condition d'arrêt anticipée, et en instanciant l'algorithme obtenu pour la fonction $F'_r(X) = G \cup \pi(X)$ (qui vérifie bien les conditions **F1**, **F2** et **F3**), nous obtenons un algorithme $\mathcal{A}(G)$ qui nous permet de calculer le point fixe interne.

Nous en déduisons l'algorithme 2 que nous appelerons STB pour *Symbolic Algorithm for Timed Büchi Games*.

Algorithme 2 L'agorithme STB (*Symbolic Algorithm for Timed Büchi Games*)

1: $Y' \leftarrow \mathsf{States}(\mathsf{Runs}(\mathcal{G}))$
2: **repeat**
3: $\quad Y \leftarrow Y'$
4: $\quad Y' \leftarrow \mathcal{A}(\mathsf{Goal} \cap \pi(Y))$
5: **until** $Y = Y'$

La correction et la terminaison de l'algorithme STB sont garanties par les résultats précédents et ceux de [58].

Remarquons que l'on a perdu l'un des gros avantages de l'algorithme SOTFBLF : la possibilité de terminaison anticipée. Néanmoins, nous gardons une certaine efficacité due à l'utilisation des zones et cette version de l'algorithme de [58] s'intègre plus naturellement avec les algorithmes précédents implantés dans Uppaal-TiGa (cf. section 3.6).

3.5.4 Jeux de sûreté avec condition de Büchi

Dans un jeu de sûreté, le joueur 1 cherche à éviter certains états « mauvais ». Dans certains systèmes, une stratégie gagnante peut être de bloquer l'évolution du système ou encore d'imposer un comportement zénon, c'est-à-dire de générer des exécutions de durée finie avec un nombre infini d'actions discrètes. Afin d'éliminer ce type de stratégies, il est possible d'ajouter

une condition de Büchi à l'objectif de sûreté, qui imposera au joueur 1 de ne pas bloquer le système.

Soit $\mathcal{G}$ une TGS et soient Goal et Bad deux ensembles d'états, respectivement à répéter et à éviter. Soit W la condition de victoire définie comme l'ensemble des exécutions ρ possédant un nombre infini d'actions discrètes et telles que $|\mathsf{States}(\rho) \cap \mathsf{Goal}| = +\infty$ et $\mathsf{States}(\rho) \cap \mathsf{Bad} = \emptyset$. Nous notons ($\mathcal{G}$,Goal,Bad) le jeu ($\mathcal{G}$,$W$).

Si Bad est une union finie de zones alors il est possible de construire très facilement un jeu de Büchi équivalent à ($\mathcal{G}$,Goal,Bad) [C3].

Supposons que $\mathsf{Bad} = \bigcup_{i\in[1..n]}(l_i,Z_i)$. Soit $\mathcal{G}'$ obtenue à partir de $\mathcal{G}$ en ajoutant :
- une nouvelle localité $B \notin \mathsf{Goal}$;
- une nouvelle action du joueur 2 err ;
- pour tout $i \in [1..n]$, une transition $(l_i, err, Z_i, \emptyset, B)$.

Comme B n'a pas de transition sortante et n'appartient pas à Goal, arriver dans B signifie pour le joueur 1 perdre le jeu de Büchi ($\mathcal{G}'$,Goal). Donc toute stratégie du joueur 1 qui passe par un état de Bad offre l'opportunité au joueur 2 d'aller dans B et donc de faire perdre le joueur 1.

Le théorème suivant formalise cette intuition :

Théorème 5 ([C3])**.** *Soient $\mathcal{G}$ une TGS et* Goal *et* Bad *deux ensembles d'états constitués d'une union finie de régions. Soit $\mathcal{G}'$ la TGS obtenue par la construction décrite ci-dessus. Une stratégie f pour le joueur 1 pour le jeu de Büchi* ($\mathcal{G}'$,Goal) *est gagnante ssi f est gagnante pour le jeu de sûreté-Büchi* ($\mathcal{G}$,Goal,Bad)*.*

Il est possible d'utiliser ce type de jeu pour éliminer les stratégies zénon du joueur 1 dans un jeu de sûreté. Soit $(\mathcal{G}, \mathsf{Goal})$ un jeu de sûreté. Supposons, que l'on mette la TGS décrite dans la figure 3.7 en parallèle de $\mathcal{G}$, nous donnant ainsi une nouvelle TGS $\mathcal{G}'$, et considérons le jeu de sûreté-Büchi ($\mathcal{G}'$,{NZ}, $\overline{\mathsf{Goal}}$).

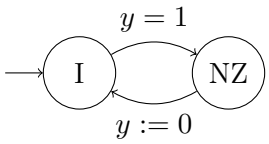


FIGURE 3.7 – Observateur pour les stratégies non zénon

Toutes les transitions de cet observateur appartiennent au joueur 1. Pour gagner, il doit satisfaire non-seulement la condition de sûreté « rester dans Goal » mais également la condition de Büchi « répéter NZ ». Il doit donc prendre les deux transitions de l'observateur infiniment souvent assurant ainsi l'écoulement du temps.

3.6 UPPAAL-TIGA

Les algorithmes présentés dans la section précédente ont été implémentés dans un outil : UPPAAL-TIGA. Né en 2005 sous la forme d'un prototype utilisant la bibliothèque de manipulation de DBM d'Alexandre David [31], cet outil a été complètement intégré dans l'environ-

nement de l'outil UPPAAL en 2006 [W3, C13], bénéficiant ainsi de nombreuses optimisations et de la richesse du langage de spécification disponibles dans UPPAAL.

L'outil permet donc la modélisation de jeux temporisés, leur simulation et la recherche de stratégies gagnantes pour des conditions de victoire d'accessibilité, de sûreté et de Büchi, ainsi que les combinaisons sûreté-accessibilité et sûreté-Büchi. Il permet également la prise en compte de jeux à observabilité partielle [C12] qui n'ont pas été présentés ici.

À la suite du calcul de l'existence d'une stratégie gagnante, UPPAAL-TIGA permet de visualiser les résultats dans le simulateur : lorsqu'il y a une stratégie gagnante, le programme propose d'essayer de la mettre en défaut (UPPAAL-TIGA joue le joueur 1 et l'utilisateur le joueur 2). Lorsqu'il n'y a pas de stratégie gagnante, il propose d'essayer de mettre en défaut l'une des stratégies qui fait perdre (UPPAAL-TIGA joue le joueur 2 et l'utilisateur le joueur 1).

Par ailleurs, UPPAAL-TIGA et ses algorithmes constituent l'un des fondements de nombreux travaux développés par la suite par Kim G. Larsen et son équipe. Citons par exemple la simulation alternante d'automates temporisés [21], la génération de tests temps réel [37, 35, 36] et une théorie d'interface pour les systèmes temps réels [32, 33, 34] (l'outil ECDAR étant une version spécialisée d'UPPAAL-TIGA).

Enfin, il faut noter au moins deux cas d'études pratiques traités avec succès en utilisant UPPAAL-TIGA : le contrôle du niveau d'huile dans une machine [24] et le contrôle du climat dans une porcherie [53].

3.7 Conclusion

Dans ce chapitre, nous avons présenté les principaux algorithmes qui se trouvent au cœur d'UPPAAL-TIGA : après avoir résolu le problème du traitement des invariants et de l'urgence, nous avons présenté des algorithmes pour la résolution des jeux temporisés d'accessibilité, de sûreté et de Büchi (ainsi qu'une combinaison de ces deux derniers).

Leur implémentation efficace dans UPPAAL-TIGA permet l'analyse de nombreux cas pratiques et fournit également une fondation solide pour de nouveaux développements basés sur les jeux temporisés.

Une perspective importante d'amélioration de ces travaux serait d'étendre l'algorithme SOTFBF pour des calculs de plus grands points fixes et de points fixes imbriqués. Cela permettrait par exemple d'obtenir un algorithme à la volée raisonnablement efficace pour la vérification de propriétés TCTL (*Timed Computation Tree Logic* [2]) avec des opérateurs imbriqués, ainsi qu'un algorithme plus efficace pour la vérification des objectifs de Büchi.

Chapitre 4

Dépliages temporisés

4.1 Introduction

4.1.1 Objectifs

L'objectif de ces travaux est de définir un nouveau dépliage symbolique pour les réseaux de Petri temporels. Le dépliage [59] donne une sémantique concurrente aux réseaux de Petri (non temporels). Cette notion peut être étendue naturellement à des produits de systèmes de transitions [42].

Dans le cadre temporisé, des travaux récents proposent des dépliages symboliques pour les réseaux d'automates temporisés [23, 15]. Concernant les réseaux de Petri temporels, une première approche a été proposée dans [27] qui génère des duplications d'événements ainsi que des arcs de lecture dans le dépliage.

Nous proposons une définition alternative du dépliage symbolique des réseaux de Petri temporels qui a notamment l'avantage d'être défini sur le dépliage du réseau non temporisé sous-jacent (section 4.3) et qui repose sur la notion locale de conflit direct. Nous montrons également comment cette approche peut être étendue aux réseaux à chronomètres (section 4.5.1) ou paramétrés (section 4.5.2) que nous avons déjà étudiés dans le cadre de sémantiques séquentielles, par exemple dans [C21, J5, C10, J2].

4.1.2 Contexte

Ces travaux ont été principalement effectués au cours de la thèse de Louis-Marie Traonouez [64] et en collaboration avec Claude Jard et Bartosz Grabiec. Nous développons ici les travaux présentés dans [64, C6, C7] et nous les avons étendus pour prendre en compte les réseaux éventuellement zénons ainsi que pour définir une notion de préfixe complet fini plus générale. Les preuves nouvelles ou modifiées se trouvent dans l'annexe B.

4.2 Réseaux de Petri et processus de branchement

4.2.1 Réseaux de Petri

Définition 22 (Réseau de Petri)**.** Un *réseau de Petri* (sauf) avec arcs de lecture est un quintuplet (P,T,F,F_r,m_0) où :

– P est un ensemble fini *places*,

– T est un ensemble fini de *transitions*, avec $P \cap T = \emptyset$,
– $F \subseteq (P \times T) \cup (T \times P)$ est la *relation de transition*,
– $F_r \subseteq P \times T$ est la *relation de lecture*,
– $m_0 \subseteq P$ est appelé le *marquage initial*.

Cette structure définit un graphe bipartite alterné et orienté tel que $(x,y) \in F \cup F_r$ ssi il existe un arc de x vers y.

On définit par ailleurs, pour tout $x \in P \cup T$, les ensembles ${}^\bullet x = \{y \in P \cup T | (y,x) \in F\}$ et $x^\bullet = \{y \in P \cup T | (x,y) \in F\}$. Pour $x \in T$, on définit également ${}^\circ x = \{y \in P | (y,x) \in F_r\}$ Ces définitions s'étendent par union aux sous-ensemble de $P \cup T$.

Pour une transition t, on dit que les places de ${}^\bullet t$ sont *consommées* par t, les places de ${}^\circ t$ sont *lues* par t, et les places de $t^\bullet$ sont *produites* par t.

Pour une place p, les transitions (et les arcs correspondants) dans ${}^\bullet p$ sont dites *en entrée* de p et les transitions dans $p^\bullet$ sont dites *en sortie* de p.

On suppose sans perte de généralité que pour toute transition t, ${}^\bullet t \neq \emptyset$. Si une transition t ne consomme aucune place, il suffit d'ajouter une nouvelle place p à P telle que $(p,t) \in F$ et $(t,p) \in F$ et d'ajouter p à m_0. Le réseau obtenu a alors un comportement totalement équivalent au réseau d'origine, même dans le cas des réseaux de Petri temporels.

Un *marquage* du réseau est un sous-ensemble de P. On dit que, dans le marquage m, $p \in P$ contient un *jeton* si $p \in m$. Une transition $t \in T$ est *sensibilisée* par le marquage m si ${}^\bullet e \cup {}^\circ e \subseteq m$. On note $\mathsf{Enabled}(m)$ l'ensemble des transitions sensibilisées par le marquage m. Une transition $t \in \mathsf{Enabled}(m)$ peut être *tirée*, conduisant à un nouveau marquage $m' = (m \setminus {}^\bullet e) \cup e^\bullet$. On note alors $m \xrightarrow{e} m'$. Un marquage m est *accessible* s'il existe une séquence finie de transitions $t_1, \ldots, t_n$ telle que $m_0 \xrightarrow{t_1} \cdots \xrightarrow{t_n} m$.

Exemple 6. La figure 4.1 présente un exemple de réseau de Petri. Les places sont représentées par des cercles, les transitions par les rectangles pleins et le marquage par les jetons dans les places : ici $m_0 = \{p_1,p_2\}$. Il y a un arc de lecture entre la place p_3 et la transition t_4 représenté par en style interrompu.

Le marquage $\{p_3,p_6\}$ est accessible, par exemple par la séquence :

$$\{p_1,p_2\} \xrightarrow{t_1} \{p_2,p_3\} \xrightarrow{t_2} \{p_3,p_4\} \xrightarrow{t_0} \{p_1,p_2\} \xrightarrow{t_2} \{p_4,p_1\} \xrightarrow{t_1} \{p_3,p_4\} \xrightarrow{t_4} \{p_3,p_6\}$$

Remarquons que la transition t_4 n'est pas possible depuis $\{p_4,p_1\}$ à cause de l'arc de lecture qui lit p_3.

4.2.2 Processus de branchement

On s'intéresse ici à une sémantique concurrente pour les réseaux de Petri par opposition à une sémantique séquentielle basée sur les systèmes de transitions, comme au chapitre 3, et qui serait définie ici par la relation $\rightarrow$ entre les marquages que nous venons de voir. La sémantique concurrente s'exprime sous la forme de *processus de branchement*.

La définition que nous proposons ici est adaptée de [42]. Les processus de branchement d'un réseau de Petri $\mathcal{N}$ sont eux-mêmes des réseaux de Petri. Dans ces réseaux de Petri, les places sont appelées *conditions* et les transitions sont appelées *événements*. Une condition est

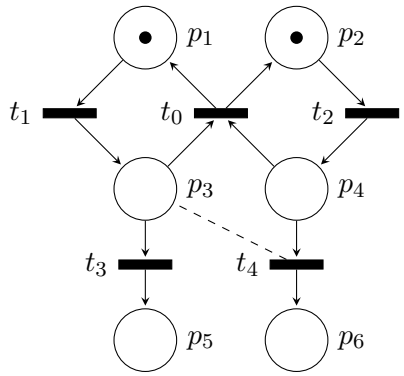


FIGURE 4.1 – Un réseau de Petri

identifiée de façon unique par le nom $(p,\{x\})$ où p est une place de $\mathcal{N}$ et x est le nom de l'unique événement en entrée. De façon similaire, un *événement* est identifié de façon unique par le nom (t,X,X') où t est une transition de $\mathcal{N}$, X est l'ensemble des noms des conditions consommées, et X' l'ensemble des noms des conditions lues.

Pour toute condition $b = (p,\{x\})$, on note $l(b) = p$. De même, pour tout événement $e = (t,X,X')$ on note $l(e) = t$. Finalement, pour un marquage m d'un processus de branchement, on note $l(m)$ le marquage de $\mathcal{N}$ tel que $p \in m$ ssi $l(p) \in l(m)$.

Définition 23 (Processus de branchement)**.** L'ensemble des *processus de branchement* d'un réseau de Petri $\mathcal{N} = (P,T,F,F_r,m_0)$ est le plus petit ensemble de réseaux de Petri satisfaisant les conditions suivantes :

1. le réseau de Petri sans transition dont l'ensemble de places et le marquage initial sont $\{(p,\emptyset)|p \in m_0\}$, est un processus de branchement de $\mathcal{N}$.
2. Soit $\mathcal{B}$ un processus de branchement de $\mathcal{N}$ et m un marquage accessible de $\mathcal{B}$ tel qu'une transition t de $\mathcal{N}$ est possible depuis le marquage $l(m)$ de $\mathcal{N}$. Soit $M = \{p \in m|l(p) \in {}^\bullet t\}$, c'est-à-dire un ensemble de conditions correspondant aux places consommées par t et soit $M' = \{p \in m|l(p) \in {}^\circ t\}$, c'est-à-dire un ensemble des conditions correspondant aux places lues par t. Soit $M'' = \{(p,\{(t,M,M')\})|p \in t^\bullet\}$. Le réseau de Petri obtenu en ajoutant à $\mathcal{B}$ les conditions de M'' et l'événement (t,M,M') avec des arcs de lecture depuis M', des arcs d'entrée depuis M et des arcs de sortie vers M'' est aussi un processus de branchement de $\mathcal{N}$. On appelle l'événement (t,M,M') une *extension possible* de $\mathcal{B}$.
3. Si $((P_i,E_i,F_i,F_{ri},m_{0i}))_i$ est une famille finie ou infinie de processus de branchement de $\mathcal{N}$ alors c'est aussi le cas de $(\bigcup_i P_i, \bigcup_i E_i, \bigcup_i F_i, \bigcup_i F_{ri}, \bigcup_i m_{0i})$.

Exemple 7. La figure 4.2 présente un processus de branchement du réseau de Petri de la figure 4.1. Nous avons omis le marquage initial qui est commun à tous les processus de branchement et n'est utile que pour leur définition.

Notons que des raccourcis ont été définis pour les noms des conditions $(b_1,b_2,\ldots)$ et des événements $(e_1,e_2,\ldots)$ afin de simplifier l'écriture. Le nom complet unique de e_5, par exemple, est :

$$\Big(t_4,\big\{(p_4,\{(t_2,\{(p_2,\emptyset)\},\emptyset)\})\big\},\big\{(p_3,\{(t_1,\{(p_1,\emptyset)\},\emptyset)\})\big\}\Big)$$

On a donc $l(e_5) = t_4$.

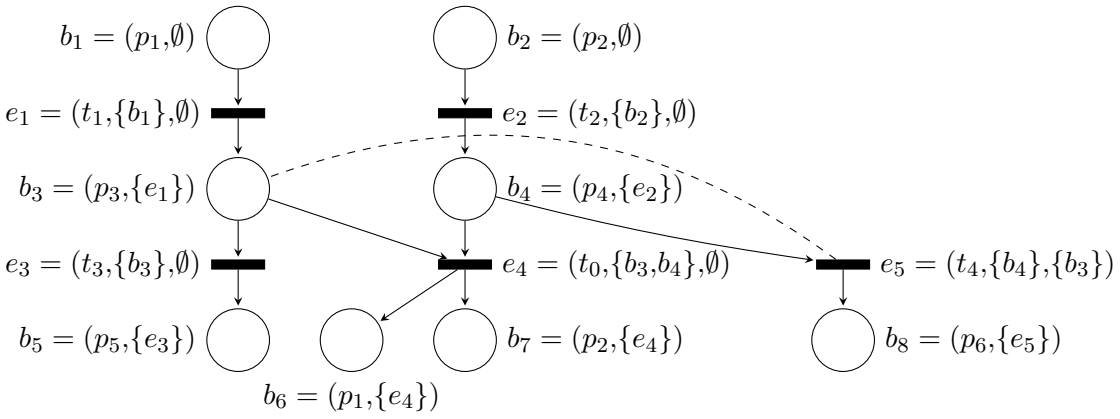


FIGURE 4.2 – Un processus de branchement du réseau de Petri de la figure 4.1

Définition 24 (Dépliage d'un réseau de Petri)**.** Le *dépliage* $\mathsf{Unfolding}(\mathcal{N})$ d'un réseau de Petri $\mathcal{N}$ est l'union de tous ses processus de branchement. On dit de tout processus de branchement qu'il est un *prefixe* de $\mathsf{Unfolding}(\mathcal{N})$.

Soit $\mathcal{B} = (B,E,F,F_r,m_0)$ un processus de branchement d'un réseau de Petri $\mathcal{N}$. On peut définir plusieurs types de relations entre les événements de E. La plus fondamentale est la causalité :

Définition 25 (Causalité)**.** La *causalité* est un ordre partiel sur $B \cup E$, noté $<$, et défini par $x < y$ ssi il existe un chemin dans $\mathcal{B}$ de x vers y avec au moins un arc dans F. On note $x \leq y$ si $x = y$ ou $x < y$.

On en déduit la notion d'histoire causale d'un événement.

Définition 26 (Histoire causale)**.** L'*histoire causale* d'un événement $e \in E$ est $\lceil e \rceil = \{e' \in E | e' \leq e\}$.

L'histoire causale de e contient tous les événements nécessaires à l'occurrence de e. Pour un ensemble d'événements $E' \subseteq E$, on définit $\lceil E' \rceil = \bigcup_{e \in E'} \lceil e \rceil$.

Exemple 8. Dans le processus de branchement de la figure 4.2, l'histoire causale de e_3 est $\lceil e_3 \rceil = \{e_1,e_3\}$ et celle de e_5 est $\lceil e_5 \rceil = \{e_1,e_2,e_5\}$.

Cependant, dans un même processus de branchement, certains événements peuvent ne pas appartenir à la même histoire. L'exemple le plus simple de ce cas de figure consiste en deux événements consommant une même condition. On parle dans ce cas de *conflit*.

Définition 27 (Ensemble d'événements en conflit)**.** Un ensemble E' d'événements est en conflit, noté $\#E'$, si :
- $\exists e_1,e_2 \in \lceil E' \rceil$ t.q. $e_1 \neq e_2$ et ${}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset$;
- ou $\exists e_1,e_2,\ldots,e_n \in \lceil E' \rceil$ t.q. $\exists i,j$ t.q. $e_i \neq e_j$ et $\forall i \in [1..n], e_i < e_{i+1}$ ou ${}^\circ e_i \cap {}^\bullet e_{i+1} \neq \emptyset$.

La définition 27 se base sur deux notions importantes pour la suite : le *conflit direct* et le *cycle de causalités faibles*. Avant de définir plus formellement ces deux notions nous définissons la notion de co-set.

Définition 28 (co-set)**.** Un co-*set* de $\mathcal{B}$ est un ensemble $B' \subseteq B$ de conditions qui sont concurrentes, c'est-à-dire sans relation causale ou de conflit : $\forall b \in B', b^\bullet \cap \lceil {}^\bullet B' \rceil = \emptyset$ et non $\#\lceil {}^\bullet B' \rceil$.

Un premier type de conflit peut survenir quand deux événements consomment la même condition, ce que nous appelons *conflit direct*.

Définition 29 (Conflit direct)**.** Deux événements e_1,$e_2 \in E$ sont en *conflit direct*, ce que nous notons $e_1 \,\mathsf{conf}\, e_2$ ssi $e_1 \neq e_2$, ${}^\bullet e_1 \cup {}^\circ e_1 \cup {}^\bullet e_2 \cup {}^\circ e_2$ est un co-set et ${}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset$.

En plus de la relation $<$, les arcs de lecture induisent un autre type de causalité dans les processus de branchement. En effet, si un événement doit lire une condition particulière qui est également consommée par un autre événement, alors la lecture doit se faire avant la consommation.

Définition 30 (Causalité faible)**.** La *causalité faible* est la relation $\nearrow$ définie par $x \nearrow y$ ssi $x < y$ ou ${}^\bullet x \cup {}^\circ x \cup {}^\bullet y \cup {}^\circ y$ est un co-set et ${}^\circ x \cap {}^\bullet y \neq \emptyset$.

La causalité faible $x \nearrow y$ indique que si x se produit alors ce doit être avant y.

Exemple 9. Dans le processus de branchement de la figure 4.2, bien que e_5 n'appartienne pas à $\lceil e_3 \rceil$, on a $e_5 \nearrow e_3$ car ${}^\circ e_5 \cap {}^\bullet e_3 \neq \emptyset$. Si ces deux événements ont lieu dans la même histoire alors e_5 a forcément lieu avant e_3.

Ainsi, un second type de conflit vient des cycles dans la relation de causalité faible : p.ex. e lit une condition consommée par e' et e' lit une condition consommée par e. Seul l'un de ces deux événements peut se produire dans une histoire donnée. Pour $X \subseteq E$, on note $\mathsf{RCycles}(X)$ l'ensemble des ensembles d'événements $\{e_1,e_2,\ldots,e_n\} \subseteq X$ tels que $e_1 \nearrow e_2 \nearrow \cdots \nearrow e_n \nearrow e_1$ et ${}^\bullet\{e_1,\ldots,e_n\}$ est un co-set.

Exemple 10. Dans le processus de branchement de la figure 4.2, $e_4 \,\mathsf{conf}\, e_5$ et donc ces deux événements ne peuvent pas avoir lieu tous les deux dans la même histoire.

Dans le processus de branchement de la figure 4.3, on a le cycle $e_1 \nearrow e_2 \nearrow e_3 \nearrow e_1$. Ces trois événements ne peuvent donc pas coexister dans une même histoire. Par contre ils le peuvent deux à deux : e_1 et e_2 peuvent coexister, de même que e_1 et e_3 ou e_2 et e_3.

Deux événements e_1 et e_2 tels que ${}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset$ peuvent ne pas être en conflit direct si leurs préconditions sont elles-mêmes en conflit. Et l'on peut faire la même remarque pour les cycles de causalités faibles. On a néanmoins le résultat suivant :

Lemme 3. *Soit E' un ensemble d'événements. $\#E'$ ssi :*
- $\exists e_1,e_2 \in \lceil E' \rceil$ *t.q.* $e_1 \,\mathsf{conf}\, e_2$ *;*
- *ou* $\mathsf{RCycles}(\lceil E' \rceil) \neq \emptyset$.

Les comportements possibles du réseau sont obtenus en résolvant les conflits des processus de branchement, donnant ainsi la notion de *configuration*, parfois aussi appelé *processus*.

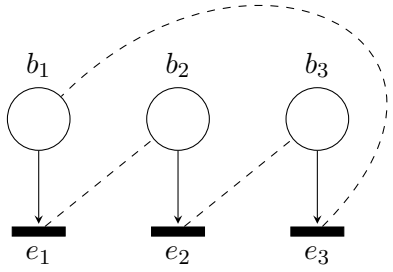


FIGURE 4.3 – Cycle de causalités faibles

Définition 31 (Configuration)**.** Une *configuration* de $\mathcal{B}$ est un sous-ensemble $C \subseteq E$ d'événements qui est *fermé par causalité* et *sans conflit*, c'est-à-dire $\forall e' \in C, \forall e \in E,\ e < e' \Rightarrow e \in C$ et non $\#C$.

On note $\mathsf{Config}(\mathcal{B})$ l'ensemble des configurations de $\mathcal{B}$.

Remarque 2. Notons que l'histoire causale $\lceil e \rceil$ de tout événement e est une configuration.

Pour toute configuration C et toute extension possible (t,M,M') de $\mathcal{B}$, on note $C \cup \{(t,M,M')\}$ l'ensemble d'événements obtenu en ajoutant (t,M,M') à C. On dit que (t,M,M') est une extension possible de C si $C \cup \{(t,M,M')\}$ est à nouveau une configuration.

Définition 32 (Coupure)**.** Une *coupure* est un co-set maximal (du point de vue de l'inclusion). Pour toute configuration C, on peut définir $\mathsf{Cut}(C)$ la coupure de C par $\mathsf{Cut}(C) = (M_0 \cup C^\bullet) \setminus {}^\bullet C$ (avec M_0 le marquage initial commun à tous les processus de branchement).

Notons que si C est finie alors $l(\mathsf{Cut}(C))$ est le marquage du réseau de Petri obtenu après la séquence d'événements dans C. Si C est infinie il s'agit seulement d'un marquage partiel constitué des places marquées n'intervenant pas infiniment souvent dans le tir des transitions correspondant aux événements de C.

Exemple 11. Dans le processus de branchement de la figure 4.2, les configurations sont les histoires causales de chaque événement plus $\emptyset$, $\{e_1,e_2\}$ et $\{e_1,e_2,e_3\}$.

Les coupures correspondent à l'application de Cut à chacune de ces configurations, ce qui inclut $\{b_3,b_8\} = \mathsf{Cut}(\lceil e_5 \rceil)$ et $\{b_1,b_2\} = \mathsf{Cut}(\emptyset)$.

Les co-sets sont les sous-ensembles non vides de ces coupures, ici tous réduits à des singletons.

4.2.3 Réseaux de Petri temporels

On note $\mathcal{I}(\mathbb{Q}_{\geq 0})$ l'ensemble des intervalles de $\mathbb{R}$ à bornes dans $\mathbb{Q}_{\geq 0} \cup \{+\infty\}$. Soit $I \in \mathcal{I}(\mathbb{Q}_{\geq 0})$ on note $I^\uparrow$ la *fermeture vers le haut* de I, c'est-à-dire le plus petit intervalle de $\mathbb{R}$ ouvert à droite en $+\infty$ et contenant I. On note de façon similaire $I^\downarrow$ la *fermeture vers le bas* de I, c'est-à-dire le plus petit intervalle de $\mathbb{R}$ ouvert à gauche en $-\infty$ et contenant I. Pour un intervalle I et un réel x on note $I + x$ l'intervalle $\{y + x \in \mathbb{R} | y \in I\}$.

Définition 33 (Réseau de Petri temporel)**.** Un *réseau de Petri temporel* sauf et avec arcs de lectures (TPN, pour *Time Petri Net*) est un sextuplet $\mathcal{N} = (P,T,F,F_r,m_0,I_s)$ tel que :

– (P,T,F,F_r,m_0) est un réseau de Petri qu'on note $\mathsf{Untimed}(\mathcal{N})$;
– $I_s : T \to \mathcal{I}(\mathbb{Q}_{\geq 0})$ associe à chaque transition une *contrainte temporelle* sous la forme d'un intervalle appelé *intervalle de tir statique* de la transition.

L'intervalle I_s restreint l'ensemble des dates auxquelles une transition peut-être tirée. La durée pendant laquelle une transition t est sensibilisée sans interruption doit se trouver dans l'intervalle $I_s(t)$ pour que t puisse tirer et ne doit jamais sortir de l'intervalle $I_s(t)^{\downarrow}$.

Exemple 12. La figure 4.4 présente une version « temporisée » du réseau de Petri de la figure 4.1. On a associé à chaque transition un intervalle temporel.

Ainsi, par exemple, la transition t_1 n'est soumise à aucune contrainte sur sa date de tir alors que t_2 ne peut tirer qu'à partir de 3 unités écoulées depuis l'arrivée d'un jeton dans p_2 et doit tirer strictement avant que 4 unités de temps se soient écoulées. La transition t_0 doit, quant à elle, tirer immédiatement dès qu'il y a des jetons dans p_3 et p_4 (à moins qu'elle ne soit désensibilisée en 0 unités de temps par le tir de t_3 ou de t_4).

Comme on le verra par la suite, pour une transition donnée, d'autre contraintes, liées aux conflits, peuvent s'ajouter à celles dûes à l'intervalle de tir statique pour la détermination des dates de tir effectives.

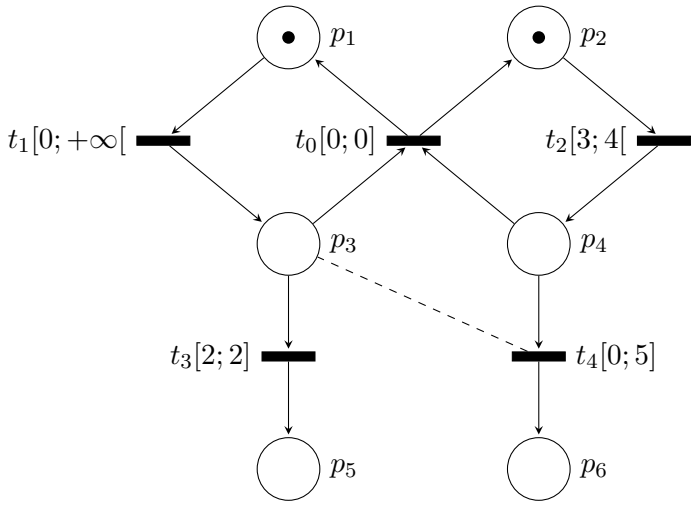


FIGURE 4.4 – Un réseau de Petri temporel

On pourrait donner la sémantique des TPN sous la forme d'un système de transitions temporisé dans lequel les états sont constitués d'un marquage et d'intervalles dynamiques [11] pour toutes les transitions ou d'horloges implicites (p.ex. [C22, 47]). Mais comme nous l'avons remarqué pour les réseaux de Petri sans temps, on obtient alors une sémantique séquentielle dans laquelle la notion de parallélisme et de concurrence a été perdue.

On donne donc la sémantique plutôt sous la forme de processus temporels [6].

Définition 34 (Processus temporel)**.** Un *processus temporel* d'un TPN $\mathcal{N}$ est un couple (C,θ) tel que :

– C est une configuration d'un processus de branchement de $\mathsf{Untimed}(\mathcal{N})$;

– $\theta : C \to \mathbb{R}_{\geq 0}$ est une fonction de temporisation donnant pour chaque événement de C une *date d'occurrence*.

Cette définition explicite comment le réseau évolue dans sa composante discrète (il se comporte comme un réseau de Petri sans temps) et à quelles dates les transitions sont tirées.

Bien sûr toutes les valeurs possibles pour θ ne respectent pas forcément les contraintes du TPN. La notion de *validité* d'un processus temporel permet de définir les processus temporels qui respectent ces contraintes, définissant ainsi une sémantique concurrente pour les TPN. Avant de définir la validité nous avons besoin de quelques notations supplémentaires.

Pour un processus temporel (C,θ) et $e \in C$ on note $\mathsf{Earlier}((C,\theta),e) = \{e' \in C | \theta(e') < \theta(e)\}$. Il s'agit de l'ensemble des événements qui se sont produits strictement avant e dans (C,θ). Lorsque (C,θ) est clair d'après le contexte, on notera simplement $\mathsf{Earlier}(e)$. On peut noter que $\mathsf{Earlier}(e)$ est toujours une configuration [6].

Soit t une transition d'un TPN $\mathcal{N}$ et $B' \subseteq C$ un co-set tel que t est sensibilisée par $l(B')$. On definit la *date de sensibilisation de t par B'* comme : $\mathsf{TOE}(C',l(e)) = \max(\{\theta(^\bullet b) | b \in B' \setminus M_0 \text{ et } l(b) \in {}^\bullet t \cup {}^\circ t\} \cup \{0\}$, avec M_0 le marquage initial de $\mathsf{Untimed}(\mathcal{N})$. C'est-à-dire que TOE vaut soit 0 si toutes les conditions consommées ou lues sont dans le marquage initial, soit la date de production de la condition consommée ou lue la plus récente sinon.

Définition 35 (Processus temporel valide)**.** Un processus temporel (C,θ) d'un TPN $\mathcal{N}$ est *valide* si pour tout $e \in C$,

$$\theta(e) - \mathsf{TOE}(^\bullet e \cup {}^\circ e, l(e)) \in I_s(l(e))^\uparrow \tag{4.1}$$

$$\forall t \in \mathsf{Enabled}(l(\mathsf{Cut}(\mathsf{Earlier}(e))), \theta(e) - \mathsf{TOE}(\mathsf{Cut}(\mathsf{Earlier}(e)),t) \in I_s(t)^\downarrow \tag{4.2}$$

Si (C,θ) est un processus temporel valide, on dit aussi que θ est une fonction de temporisation *valide* de C.

L'équation 4.1 impose que la durée de sensibilisation de la transition tirée ait dépassé la borne inférieure de l'intervalle statique de tir et l'équation 4.2 impose que les durées de sensibilisation de toutes les transitions sensibilisées du réseau ne dépassent la borne supérieure de leur intervalle statique de tir.

Cette dernière condition est source de difficultés car elle n'est pas locale à une transition donnée. Ainsi la date de tir d'une transition peut être contrainte, en plus de l'intervalle statique de tir, par une autre transition sensibilisée n'importe où dans le réseau.

Pour un processus temporel (C,θ) et $C' \subseteq C$, on note de façon légèrement abusive (C',θ) le processus temporel composé de C' et de la restriction de θ à C'.

Exemple 13. La figure 4.5 présente deux processus temporels valides du TPN de la figure 4.4. On rappelle la valeur de $l(e)$ à côté de chaque événement e, puis on donne la date d'occurrence de e. Les étiquettes des événements ont donc la forme $e; l(e); \theta(e)$. De même les étiquettes des conditions ont la forme $b; l(b)$.

Dans le processus temporel de la figure 4.5a, on aurait aussi pu ajouter l'événement e_3 à la date 3,52 au lieu de e_4 (les deux événements étant en conflit).

Dans le processus temporel de la figure 4.5b, par contre, il n'est pas possible de remplacer e_3 par e_4, à n'importe quelle date. En effet, avec 0,13 comme date d'occurrence pour e_1, e_3 a forcément lieu avant e_2 qui se trouve dans l'histoire causale de e_4.

De même, dans le processus temporel de la figure 4.5a, pour toute date d'occurrence de e_2 dans l'intervalle $[3; 3{,}52[$, l'occurrence de e_3 n'est pas possible. On voit donc que l'occurrence

de e_3 dépend non seulement de son passé causal $\{e_1\}$ mais également d'autres événements comme e_2 qui ne s'y trouvent pas. Cela est dû au conflit entre e_3 et e_4 qui « transfère » certaines contraintes de e_4 vers e_3 (et inversement).

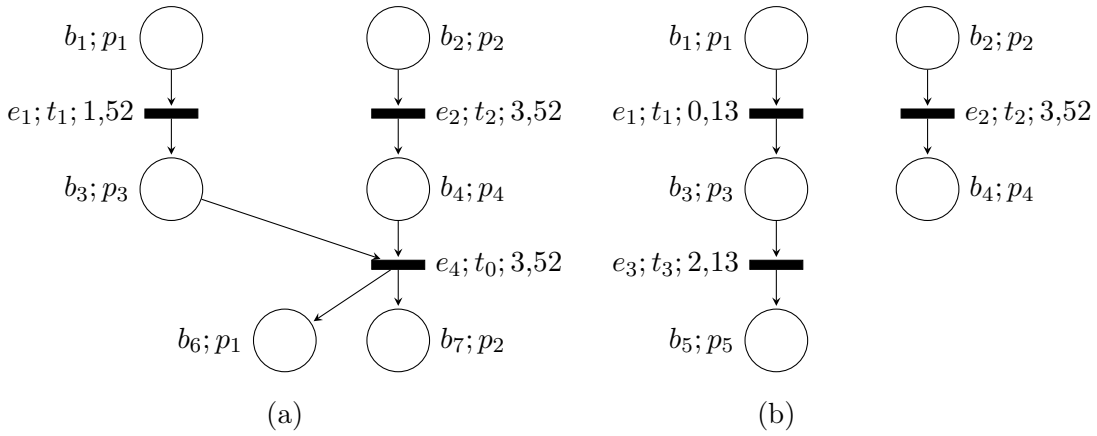


FIGURE 4.5 – Deux processus temporels valides du TPN de la figure 4.4

4.3 Processus de branchement temporels

Comme nous venons de le voir, les processus temporels d'un TPN sont des temporisations des *configurations* du réseau de Petri non temporel sous-jacent et non des temporisations des processus de branchement. Cela est dû au fait que des événements en conflit étant mutuellement exclusifs, il n'est pas possible de leur associer simultanément des dates d'occurrence.

Pour permettre la temporisation des processus de branchement, nous proposons donc d'étendre les fonctions de temporisation en permettant d'associer la date d'occurrence $+\infty$ à un événement qui n'a pas lieu. On obtient alors la notion suivante de *processus de branchement temporel*.

Définition 36 (Processus de branchement temporel)**.** Un *processus de branchement temporel* (TBP pour *Time Branching Process*) d'un TPN $\mathcal{N}$ est un couple $(\mathcal{B},\theta)$ tel que :
- $\mathcal{B} = (B,E,F,F_r,m_0)$ est un processus de branchement de $\mathsf{Untimed}(\mathcal{N})$;
- $\theta : E \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ est une fonction de temporisation donnant pour chaque événement de $\mathcal{B}$ une *date d'occurrence*.

Comme précédemment, nous donnons un certain nombre de contraintes pour qu'un processus de branchement temporel reflète le comportement du TPN, et qui définissent les processus de branchement temporels *valides*.

Définition 37 (Processus de branchement temporel valide)**.** Un processus de branchement temporel $(\mathcal{B},\theta)$ d'un TPN $\mathcal{N}$, avec E l'ensemble d'événements de $\mathcal{B}$, est *valide* si

$$\forall E' \in \mathsf{RCycles}(E), \exists e' \in E' \text{ t.q. } \theta(e') = +\infty \tag{4.3}$$

et $\forall e \in E$:

$$\Big[\big[\theta(e) \neq +\infty \text{ et } \theta(e) - \mathsf{TOE}({}^\bullet e \cup {}^\circ e, l(e)) \in I_s(l(e)) \tag{4.4}$$

$$\text{et } \forall e' \in E \text{ t.q. } e \,\mathsf{conf}\, e', \theta(e') = +\infty \tag{4.5}$$

$$\text{et } \forall e' \in E \text{ t.q. } e \nearrow e', \theta(e) \leq \theta(e')\big] \tag{4.6}$$

$$\text{ou } \big[\theta(e) = +\infty \text{ et } \exists b \in {}^\bullet e \cup {}^\circ e, \theta({}^\bullet b) = +\infty\big] \tag{4.7}$$

$$\text{ou } \big[\theta(e) = +\infty \text{ et } \exists e' \in E \text{ t.q. } (e \,\mathsf{conf}\, e' \text{ ou } e \nearrow e') \text{ et } \theta(e') \neq +\infty \text{ et } \theta(e') - \mathsf{TOE}({}^\bullet e \cup {}^\circ e, l(e)) \in I_s(l(e))^{\downarrow}\big]\Big] \tag{4.8}$$

L'équation 4.4 indique que chaque transition doit tirer lorsque sa durée de sensibilisation appartient à son intervalle de tir statique.

L'équation 4.3 indique qu'en cas de cycle dans la relation de causalité faible, un des événements impliqués au moins ne se produit pas.

L'équation 4.5 indique que si deux événements sont en conflit direct alors l'un des deux ne se produit pas.

L'équation 4.6 indique que s'il y a une causalité faible entre deux événements, si l'événement lecteur se produit, c'est nécessairement avant l'événement consommateur. Le cas où la causalité est également forte est redondant avec la contrainte de l'équation 4.4.

L'équation 4.7 indique que si un événement ne se produit pas alors aucun de ses successeurs causaux ne se produit.

Finalement, l'équation 4.8 indique que si un événement ne se produit pas, c'est qu'un événement en conflit direct (ou consommateur d'une condition à lire) s'est produit avant qu'il n'ait été obligé de se produire à cause de la borne supérieure de l'intervalle de tir statique.

Exemple 14. L'ensemble des contraintes sur les fonctions de temporisation du processus de branchement de la figure 4.2, pour le TPN de la figure 4.4 est donné par :

- pour e_1 :
$$0 \leq \theta(e_1)$$
- pour e_2 :
$$3 \leq \theta(e_2) < 4$$
- pour e_3 :
$$\theta(e_3) = +\infty \text{ et } \theta(e_4) \leq \theta(e_1) + 2$$
$$\text{ou}$$
$$\theta(e_3) = \theta(e_1) + 2 \text{ et } \theta(e_4) = +\infty$$
- pour e_4 :
$$\theta(e_4) = +\infty \text{ et } (\theta(e_3) \leq \max\{\theta(e_1),\theta(e_2)\} \text{ ou } \theta(e_5) \leq \max\{\theta(e_1),\theta(e_2)\})$$
$$\text{ou}$$
$$\theta(e_4) = \max\{\theta(e_1),\theta(e_2)\} \text{ et } \theta(e_3) = +\infty \text{ et } \theta(e_5) = +\infty$$
- pour e_5 :
$$\theta(e_5) = +\infty \text{ et } (\theta(e_3) \leq \max\{\theta(e_1),\theta(e_2)\} + 5 \text{ ou } \theta(e_4) \leq \max\{\theta(e_1),\theta(e_2)\} + 5)$$
$$\text{ou}$$
$$\max\{\theta(e_1),\theta(e_2)\} \leq \theta(e_5) \leq \max\{\theta(e_1),\theta(e_2)\} + 5 \text{ et } \theta(e_4) = +\infty \text{ et } \theta(e_5) \leq \theta(e_3)$$

Après simplification, on obtient :

$$\left\{\begin{array}{l} 1 \leq \theta(e_1) \\ 3 \leq \theta(e_2) < 4 \\ \theta(e_3) = +\infty \\ \theta(e_4) = \max\{\theta(e_1),\theta(e_2)\} \\ \theta(e_5) = +\infty \end{array}\right. \text{ ou } \left\{\begin{array}{l} 0 \leq \theta(e_1) < 2 \\ 3 \leq \theta(e_2) < 4 \\ \theta(e_3) = \theta(e_1) + 2 \\ \theta(e_3) \leq \theta(e_2) \\ \theta(e_4) = +\infty \\ \theta(e_5) = +\infty \end{array}\right. \text{ ou } \left\{\begin{array}{l} 0 \leq \theta(e_1) < 2 \\ 3 \leq \theta(e_2) < 4 \\ \theta(e_3) = \theta(e_1) + 2 \\ \theta(e_3) = \theta(e_2) \\ \theta(e_4) = +\infty \\ \theta(e_5) = \theta(e_2) \end{array}\right.$$

On distingue encore, parmi les processus de branchement temporels, ceux qui sont dits *temporellement complets*. Il s'agit de ceux pour lesquels aucune composante parallèle n'est « en retard », c'est-à-dire qu'on ne peut étendre le processus de branchement avec un événement qui aurait forcément lieu avant l'un de ceux déjà présent.

Définition 38 (TBP temporellement complet)**.** Soit $(\mathcal{B},\theta)$ un processus de branchement temporel d'un réseau de Petri temporel $\mathcal{N}$. Soit E l'ensemble des événements de $\mathcal{B}$. $(\mathcal{B},\theta)$ est *temporellement complet* si pour toute extension (t,M,M') :

- soit il existe $e \in E$ avec $\theta(e) \neq +\infty$ tel que ${}^\bullet e \cap (M \cup M') \neq \emptyset$ et $\theta(e) - \mathsf{TOE}(M \cup M',t) \in I_s(t)^\downarrow$,
- soit $\max\{\theta(e) | e \in E, \theta(e) \neq +\infty\} - \mathsf{TOE}(M \cup M',t) \in I_s(t)^\downarrow$

Exemple 15. Le processus temporel de la figure 4.5b est également un processus de branchement temporel. Il est temporellement complet. Cependant, son préfixe obtenu en supprimant l'événement e_3 (et la condition b_5) ne l'est pas car toutes les dates d'occurrence possibles pour e_3 (ici le singleton $\{2{,}13\}$) sont inférieures à celle de e_2.

La correction et la complétude de la description des comportements d'un TPN sous la forme de processus de branchement temporels valides sont établies respectivement par les théorèmes 6 et 7.

Théorème 6. *Soit $\mathcal{N}$ un réseau de Petri temporel. Soit $(\mathcal{B},\theta)$ un processus de branchement temporel valide temporellement complet de $\mathcal{N}$. Soit $E_{<+\infty} = \{e \in E \ t.q. \theta(e) < \infty\}$ et $\theta_{<+\infty}$ la restriction de θ à $E_{<+\infty}$.*

$(E_{<+\infty},\theta_{<+\infty})$ est un processus temporel valide de $\mathcal{N}$.

Théorème 7. *Soit $\mathcal{N}$ un réseau de Petri temporel. Soit (C,θ) un processus temporel valide de $\mathcal{N}$. C est une configuration du processus de branchement $\mathcal{B} = (C^\bullet,C,F,F_r,m_0)$ de $\mathsf{Untimed}(\mathcal{N})$.*

$(\mathcal{B},\theta)$ est un processus de branchement temporel valide temporellement complet.

Nous pouvons maintenant définir la notion de dépliage *symbolique* d'un réseau de Petri temporel de la même façon que nous avions défini le dépliage d'un réseau de Petri (non temporel).

Définition 39 (Dépliage symbolique)**.** Le *dépliage symbolique* d'un réseau de Petri temporel $\mathcal{N}$, noté $\mathsf{Unfolding}(\mathcal{N})$ est l'union de tous les processus de branchement temporels valides temporellement complets de $\mathcal{N}$.

De façon équivalente, le dépliage symbolique est également obtenu par l'union de tous les processus de branchement temporels valides dont le processus de branchement *sous-jacent* est $\mathsf{Unfolding}(\mathsf{Untimed}(\mathcal{N}))$ (qui est trivialement temporellement complet). Cela nous amène à la notion de processus de branchement temporel symbolique :

Définition 40 (Processus de branchement temporel symbolique)**.** Un *processus de branchement temporel symbolique* est l'union de tous les processus de branchement temporels valides temporellement complets ayant le même processus de branchement (non temporel) sous-jacent. On le note $(\mathcal{B},D)$ où $\mathcal{B}$ est le processus de branchement sous-jacent commun et D est l'union des fonctions de temporisations valides associées.

On retrouve alors un résultat habituel pour les représentations symboliques de systèmes temporisés :

Proposition 3. *L'ensemble D des fonctions de temporisation d'un processus de branchement temporel symbolique fini (dont le processus de branchement sous-jacent est fini) $(\mathcal{B},D)$ peut être représenté par une union finie de zones à coefficients rationnels.*

4.4 Préfixe complet fini

Le dépliage d'un réseau de Petri est, en général, infini. Nous pouvons cependant toujours en extraire un préfixe fini complet, c'est-à-dire qui permet de retrouver le dépliage infini. Nous montrons maintenant comment faire la même chose avec le dépliage symbolique d'un TPN.

4.4.1 Processus temporels futur-équivalents

Nous nous intéressons d'abord à une certaine notion d'équivalence entre les processus temporels basée sur la notion d'âge réduit des conditions déjà introduite, par exemple, dans [27].

Définition 41 (Âge réduit d'une condition)**.** Pour tout co-set A, toute fonction de temporisation θ et toute condition $b \in A$, l'*âge (réduit)* de b dans A est :

$$\mathsf{age}(b,\theta,A) = \min\{\max_{b' \in A}\{\theta(^\bullet b')\} - \theta(^\bullet b), K^*(b)\}$$

$$\text{avec } K^*(b) = \max\{K(t) \text{ t.q. } t \in T \text{ et } t \in l(b)^\bullet\}$$

$$\text{et } K(t) = \begin{cases} \max I_s(t) \text{ si } \max I_s(t) \neq +\infty \\ \min I_s(t) \text{ sinon} \end{cases}$$

Nous arrivons alors à la notion de processus temporels *futur-équivalents* c'est-à-dire qui vont se comporter de façon identique dans le futur.

Définition 42 (Processus temporels futur-équivalents)**.** Soient (C,θ) et (C',θ') deux processus temporels d'un TPN $\mathcal{N}$. (C,θ) et (C',θ') sont *futur-équivalents* si
- $l(\mathsf{Cut}(C)) = l(\mathsf{Cut}(C'))$
- $\forall b \in \mathsf{Cut}(C), \forall b' \in \mathsf{Cut}(C')$, t.q. $l(b) = l(b'), \mathsf{age}(b,\theta,\mathsf{Cut}(C)) = \mathsf{age}(b',\theta',\mathsf{Cut}(C'))$.

Exemple 16. Le processus temporel de la figure 4.5a et le processus temporel ne contenant aucun événement sont futur-équivalents : $l(b_6) = l(b_1)$, $l(b_7) = l(b_2)$ et toutes ces conditions ont un âge réduit nul.

Le théorème 8 montre que deux processus temporels futur-équivalents peuvent effectivement générer les mêmes comportements.

Soient (C,θ) un processus temporel et (t,M,M_r) une extension possible de C. On note (C_t,θ_δ) le processus temporel obtenu en ajoutant t à C à la date δ, c'est-à-dire $C_t = C \cup \{(t,M,M_r)\}$ et $\forall e \in C, \theta_\delta(e) = \theta(e)$ et $\theta_\delta((t,M,M_r)) = \delta$.

Théorème 8. *Soient* (C,θ) *et* (C',θ') *deux processus temporels valides et futurs-équivalents d'un TPN* $\mathcal{N}$. *Soit* (t,M,M_r) *une extension possible de* C *telle qu'il existe une extension possible* (t,M',M'_r) *de* C'.

Alors pour tout $\delta \geq \max_{b\in\mathsf{Cut}(C)} \theta({}^\bullet b)$ *tel que* (C_t,θ_δ) *est valide,* $(C'_t,\theta'_{\delta'})$ *est valide pour* $\delta' = \delta - \max_{b\in\mathsf{Cut}(C)} \theta({}^\bullet b) + \max_{b\in\mathsf{Cut}(C')} \theta'({}^\bullet b)$.

De plus, (C_t,θ_δ) *et* $(C'_t,\theta'_{\delta'})$ *sont futur-équivalents.*

4.4.2 Événements d'arrêt et préfixe fini complet

Notons que dans le théorème 8, en l'absence d'arcs de lecture, l'existence de l'extension (t,M',M'_r) de C' est garantie par le fait que, les deux processus étant futurs-équivalents, on a $l(\mathsf{Cut}(C)) = l(\mathsf{Cut}(C'))$ [43]. Ce n'est plus le cas en présence d'arcs de lecture [66] et, ces derniers introduisent ainsi des complications dans le calcul du préfixe complet qui, si l'esprit est le même que pour la construction originelle, rendraient la présentation peu lisible [8]. Dans cette section, on suppose donc, pour simplifier, qu'il n'y a pas d'arcs de lecture.

Nous définissons maintenant les événements à partir desquels il est possible d'arrêter le dépliage du réseau sans perte d'information. On se base pour cela sur la notion d'*ordre adéquat* [43].

Définition 43 (Ordre adéquat)**.** Une relation binaire $\prec$ sur l'ensemble des configurations finies est un ordre adéquat si :

- $\prec$ est irréflexive et transitive ;
- $\prec$ raffine l'ordre préfixe : $C \subset C'$ implique $C \prec C'$;
- $\prec$ est préservée par extension : si $\mathsf{Cut}(C) = \mathsf{Cut}(C')$ alors pour toute extension possible (t,M) de C et l'extension correspondante (t,M') de C', $C \prec C'$ implique $C \cup \{(t,M)\} \prec C' \cup \{(t,M')\}$.

Chatain et Khomenko ont en outre prouvé que tout ordre adéquat est bien fondé [28] : tout ensemble non vide de configurations finies a un plus petit élément pour $\prec$.

Soit e un événement. Notons $\mathsf{Past}((C,\theta),e) = \lceil e \rceil \cup \mathsf{Earlier}((C,\theta),e)$. Cet ensemble représente tous les événements de (C,θ) qui se sont nécessairement déjà produits quand e a lieu.

Notons $\mathcal{P}(e)$ l'ensemble des processus temporels finis et valides de $\mathsf{Unfolding}(\mathcal{N})$ qui contiennent e.

Définition 44 (Événement d'arrêt)**.** Soit $\mathcal{N}$ un TPN et $\prec$ un ordre adéquat sur les configurations de $\mathcal{N}$.

Un événement e de $\mathsf{Unfolding}(\mathsf{Untimed}(\mathcal{N}))$ est un *événement d'arrêt* (*Cut-off event*) du dépliage symbolique $\mathsf{Unfolding}(\mathcal{N})$ s'il existe un autre événement e' de $\mathsf{Unfolding}(\mathsf{Untimed}(\mathcal{N}))$ tel que :

- $\lceil e' \rceil \prec \lceil e \rceil$;
- $l(\mathsf{Cut}(\lceil e' \rceil)) = l(\mathsf{Cut}(\lceil e \rceil))$;
- Pour tout $(C,\theta) \in \mathcal{P}(e)$, il existe $(C',\theta') \in \mathcal{P}(e')$ tel que $(\mathsf{Past}((C,\theta),e),\theta)$ et $(\mathsf{Past}((C',\theta'),e'),\theta')$ sont futur-équivalents.

La cohérence de la définition 44 est assurée par la proposition suivante :

Proposition 4. *Soit* $\mathcal{N}$ *un TPN. Pour tout processus temporel valide* (C,θ) *de* $\mathcal{N}$ *et tout événement* $e \in C$, $(\mathsf{Past}((C,\theta),e),\theta)$ *est un processus temporel valide de* $\mathcal{N}$.

Il est facile de voir que ce résultat n'est pas vrai pour $\lceil e \rceil$: $(\lceil e \rceil, \theta)$ n'est pas forcément valide car des événements non causalement reliés à e peuvent tout de même être forcés de survenir strictement avant e ce qui contredit la contrainte de l'équation 4.2.

Il est par ailleurs intéressant de remarquer que si les intervalles statiques sont tous [0,0] ou tous $[0, +\infty[$, on retrouve la notion d'événement d'arrêt de [43], c'est-à-dire les deux premières conditions uniquement.

En effet, dans le premier cas, la seule temporisation valide de toute configuration C consiste à mettre une date d'occurrence nulle à tous les événements. Donc pour tout processus temporel et tout événement e, $\mathsf{Earlier}(e) = \emptyset$. Donc $\mathsf{Past}(e) = \lceil e \rceil$ et puisque les âges réduits des conditions sont tous nuls, la deuxième condition de la définition 44 implique la troisième.

Dans le second cas, pour toute configuration C, la validité d'une fonction de temporisation θ se réduit à $\forall e \in C, \theta(e) \geq \mathsf{TOE}({}^\bullet e, l(e))$, ce qui est équivalent à $e' < e \Rightarrow \theta(e') \leq \theta(e)$. On sait grâce à la deuxième condition que $\lceil e' \rceil$ s'étend en une configuration C' telle que $l(\mathsf{Cut}(\lceil C' \rceil)) = l(\mathsf{Cut}(\lceil C \rceil))$. Soient $b_1, \ldots, b_n$ les conditions de $\mathsf{Cut}(C')$ rangées par dates de production croissantes et $b'_1, \ldots, b'_n$ les conditions de $\mathsf{Cut}(C')$ telles que $\forall i, l(b_i) = l(b'_i)$. Pour tout i, on donne à tous les événements de $\lceil {}^\bullet b'_i \rceil$ sauf ${}^\bullet b'_i$ la même date d'occurrence que leur prédécesseur le plus récent. Et pour ${}^\bullet b'_i$ on donne la date d'occurence de ${}^\bullet b_i$, qui par définition de l'ordre sur les b_i est bien supérieure ou égale à toutes les dates des autres événements de $\lceil {}^\bullet b'_i \rceil$. On obtient ainsi un processus temporel (C', θ') valide et futur-équivalent à (C, θ). Là encore la deuxième condition de la définition 44 implique donc la troisième.

Finalement, nous définissons un préfixe du dépliage symbolique qui ne contient aucun événement d'arrêt. Le théorème 9 montre que ce préfixe est complet et le théorème 10 qu'il est fini.

Définition 45 (Préfixe maximal sans événement d'arrêt)**.** Soit $\mathcal{N}$ un TPN. Le *préfixe maximal sans événement d'arrêt relativement à* $\prec$ de $\mathsf{Unfolding}(\mathcal{N})$, noté $CFP_\prec(\mathcal{N})$ est, au sens de l'inclusion des ensemble d'événements, le plus grand préfixe de $\mathsf{Unfolding}(\mathcal{N})$ qui ne contient aucun événement d'arrêt.

Théorème 9. *Soit $\mathcal{N}$ un TPN. Pour tout processus temporel fini et valide (C,θ) de $\mathsf{Unfolding}(\mathcal{N})$ il existe un processus temporel fini et valide (C',θ') dans $CFP_\prec(\mathcal{N})$ tel que (C,θ) et (C',θ') sont futur-équivalents.*

Théorème 10. *Soit $\mathcal{N}$ un TPN. $CFP_\prec(\mathcal{N})$ est fini.*

4.5 Réseaux de Petri temporels étendus

Nous montrons maintenant rapidement comment étendre ces résultats au cas des réseaux de Petri à chronomètres et des réseaux de Petri à chronomètres paramétrés.

4.5.1 Chronomètres

L'introduction de chronomètres, c'est-à-dire d'horloges dont l'évolution peut être temporairement « gelée », permet la modélisation de systèmes complexes. L'utilisation la plus emblématique est peut-être celle de la modélisation d'ordonnanceurs temps réels préemptifs : les chronomètres permettent ainsi de modéliser l'état d'avancement des tâches en tenant compte des préemptions.

Nous considérons ici le modèle introduit dans [J5] à base d'arcs activateurs. En effet, les arcs activateurs se traduiront au niveau du dépliage par des arcs de lecture, ce qui nous permet de rentrer dans le cadre exposé précédemment.

Définition 46 (Réseau de Petri à chronomètres)**.** Un *réseau de Petri à chronomètres* (*Stopwatch Petri Net*, SwPN) sauf et avec arcs activateurs est un septuplet $\mathcal{N} = (P,T,F,F_r,F_a,m_0,I_s)$ avec :

- (P,T,F,F_r,m_0,I_s) est un réseau de Petri *temporel* ;
- $F_a \subseteq P \times T$ est la relation d'*activation*.

Comme précédemment, pour toute transition $t \in T$, nous notons ${}^{\diamond}t = \{p \in P | (p,t) \in F_a\}$ l'ensemble des places *activatrices* de t.

Pour un marquage m, une transition t est dite active si ${}^{\bullet}t \cup {}^{\circ}t \cup {}^{\diamond}t \subseteq m$, c'est-à-dire si t est sensibilisée par m et que toutes ses places activatrices sont marquées. On note $\mathsf{Active}(m)$ l'ensemble des transitions qui sont actives dans le marquage m.

Intuitivement, un SwPN fonctionne comme un TPN en remplaçant la notion de sensibilisation par celle d'activité : une transition peut tirer si elle est active et que sa *durée d'activité* est comprise dans l'intervalle de tir statique. Quand une transition sensibilisée t est liée a une place p par la relation d'activation, la durée d'activité de la transition augmente *que* lorsque p est marquée. p n'intervient cependant pas dans la sensibilisation de t. Quand t est sensibilisée mais que p n'est pas marquée, la durée d'activité de la transition est « gelée » : elle garde sa valeur mais n'évolue plus bien que globalement le temps s'écoule. Quand t n'est pas sensibilisée cette durée est nulle.

Exemple 17. La figure 4.6 présente un exemple de réseau de Petri à chronomètres. Il possède un unique arc activateur entre p_2 et t_1 distingué par une extrémité en forme de losange.

Ainsi, la durée d'activité de t_1 ne progresse que que lorsque p_2 est marquée et il faudra donc, au moins, la séquence t_2,t_3,t_2,t_3 avant que t_1 ne puisse tirer. Cette séquence permet en effet, au plus, l'augmentation de la durée d'activité de t_1 de 8 unités de temps. On pourra ensuite écouler jusqu'à 4 unités de temps avant que t_2 ne soit forcée à tirer à nouveau, ce qui nous permet d'atteindre les 10 unités de temps requises par t_1.

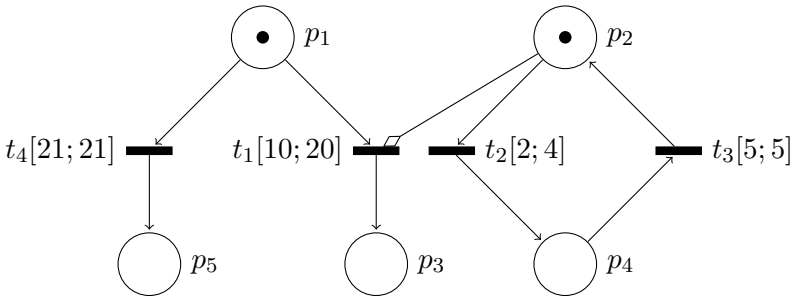


FIGURE 4.6 – Un réseau de Petri à chronomètres

Dans les TPN, pour un processus temporel (C,θ) la durée de sensibilisation d'une transition t par les conditions contenues dans le co-set B à la date τ est donnée par $\tau - \mathsf{TOE}(B,t)$. Ce sont ces durées qui sont contraintes par la notion de validité des processus temporels.

Pour les SwPN, la durée de sensibilisation doit être remplacée dans ces contraintes par la durée d'activité, c'est-à-dire la somme des durées pendant lesquelles la transition est active.

Nous donnons comme précédemment une sémantique concurrente aux SwPN sous la forme de processus temporels.

Pour une SwPN $\mathcal{N} = (P,T,F,F_r,F_a,m_0,I_s)$, notons $\mathsf{Underlying}(\mathcal{N})$ le réseau de Petri avec arcs de lecture $(P,T,F,F_r \cup F_a,m_0)$. Remarquons que les arcs d'activations de $\mathcal{N}$ se transforment en arcs de lecture dans $\mathsf{Underlying}(\mathcal{N})$, ce qui traduit la dépendance entre une transition et les places qui l'activent.

Nous pouvons alors définir les processus temporels d'un SwPN $\mathcal{N}$:

Définition 47 (Processus temporel d'un SwPN)**.** Un *processus temporel* d'un SwPN $\mathcal{N}$ est un couple (C,θ) tel que :

- C est une configuration d'un processus de branchement de $\mathsf{Underlying}(\mathcal{N})$;
- $\theta : C \to \mathbb{R}_{\geq 0}$ est une fonction de temporisation donnant pour chaque événement de C une *date d'occurrence*.

Soit un processus temporel (C,θ) d'un SwPN $\mathcal{N}$.

La *durée* d'un co-set B dans (C,θ), à la date τ est :

$$\mathsf{Duration}(B,\tau) = \max\{\min\{\min_{e \in B^\bullet}\{\theta(e)\},\tau\} - \max_{b \in B}\{\theta({}^\bullet b)\}\}$$

Il s'agit de la date de la première consommation de l'une des conditions de B moins la date de la dernière production de l'une des conditions de B. Entre ces deux dates, le co-set existe : toutes ses conditions ont été produites et pas encore consommées.

Pour une transition t est un co-set B tel que $l(B) = {}^\bullet t \cup {}^\circ t$, on note $\mathsf{Acosets}(B,t)$ l'ensemble des co-sets $B \cup B'$ tels que $l(B') = {}^\diamond t$.

La *durée d'activitée* à la date τ d'une transition t par le co-set B est alors :

$$\mathsf{DOA}(B,t,\tau) = \sum_{A \in \mathsf{Acosets}(B,t)} \mathsf{Duration}(A,\tau)$$

Pour un co-set A, on note $E(A,t) = \{b \in A | l(b) \in {}^\bullet t \cup {}^\circ t\}$ l'ensemble des conditions de A qui sensibilisent t et ${}^\star e = E({}^\bullet e \cup {}^\circ e, l(e))$, les conditions consommées ou lues par e et qui ne correspondent à des places activatrices.

La validité d'un processus temporel est alors donnée par la définition suivante :

Définition 48 (Processus temporel valide d'un SwPN)**.** Un processus temporel (C,θ) d'un SwPN $\mathcal{N}$ est *valide* si pour tout $e \in C$,

$$\theta(e) \geq \max_{b \in {}^\bullet e \cup {}^\circ e} \theta({}^\bullet b) \tag{4.9}$$

$$\mathsf{DOA}({}^\star e, l(e), \theta(e)) \in I_s(l(e))^\uparrow \tag{4.10}$$

$$\forall t \in \mathsf{Active}(l(\mathsf{Cut}(\mathsf{Earlier}(e))), \mathsf{DOA}(E(\mathsf{Cut}(\mathsf{Earlier}(e)),t),t,\theta(e)) \in I_s(t)^\downarrow \tag{4.11}$$

Contrairement au cas des TPN, on doit assurer explicitement un écoulement du temps qui respecte la causalité avec l'équation 4.9 qui n'est ici pas une conséquence de l'équation 4.10.

Les processus de branchement temporels d'un SwPN sont définis de façon similaire :

Définition 49 (Processus de branchement temporel d'un SwPN)**.** Un *processus de branchement temporel* d'un SwPN $\mathcal{N}$ est un couple $(\mathcal{B},\theta)$ tel que :

- $\mathcal{B} = (B,E,F,F_r,m_0)$ est un processus de branchement de $\mathsf{Underlying}(\mathcal{N})$;
- $\theta : E \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ est une fonction de temporisation donnant pour chaque événement de $\mathcal{B}$ une *date d'occurrence*.

Définition 50 (Processus de branchement temporel valide d'un SwPN)**.** Un processus de branchement temporel $(\mathcal{B},\theta)$ d'un SwPN $\mathcal{N}$, avec E l'ensemble d'événements de $\mathcal{B}$, est *valide* si

$$\forall E' \in \mathsf{RCycles}(E), \exists e' \in E' \text{ t.q. } \theta(e') = +\infty \tag{4.12}$$

et $\forall e \in E$:

$$\Big[\big[\theta(e) \neq +\infty \text{ et } \theta(e) \geq \max_{b \in {}^\bullet e \cup {}^\circ e} \theta({}^\bullet b) \tag{4.13}$$

$$\text{et } \mathsf{DOA}({}^\star e,l(e),\theta(e)) \in I_s(l(e)) \tag{4.14}$$

$$\text{et } \forall e' \in E \text{ t.q. } e \,\mathsf{conf}\, e', \theta(e') = +\infty \tag{4.15}$$

$$\text{et } \forall e' \in E \text{ t.q. } e \nearrow e', \theta(e) \leq \theta(e')\big] \tag{4.16}$$

$$\text{ou } \big[\theta(e) = +\infty \text{ et } \exists b \in {}^\bullet e \cup {}^\circ e, \theta({}^\bullet b) = +\infty\big] \tag{4.17}$$

$$\text{ou } \big[\theta(e) = +\infty \text{ et } \exists e' \in E \text{ t.q. } (e \,\mathsf{conf}\, e' \text{ ou } e \nearrow e') \text{ et } \theta(e') \neq +\infty \text{ et } \mathsf{DOA}({}^\star e, l(e),\theta(e')) \in I_s(l(e))^{\downarrow}\big]\Big] \tag{4.18}$$

Notons que conf et $\nearrow$ s'interprètent sur $\mathsf{Underlying}(\mathcal{N})$ et concernent donc également les arcs de lecture induits par les arcs activateurs.

Exemple 18. La figure 4.7 présente un processus de branchement temporel valide du SwPN de la figure 4.6.

Remarquons en particulier les événements e_1 et e_4 qui correspondent à des occasions possibles de tirer t_1 « ratées » car le jeton de p_2 a été consommé à chaque fois par t_2. Mais de toute façon, la durée d'activité de t_1 était inférieure à 10 et donc t_1 n'était pas tirable.

En effet, au moment où a lieu e_2, rendant t_1 inactive, on a $\mathsf{DOA}(\{b_1\},t_1,\theta(e_2)) = 3{,}2$, ce qui est bien inférieur à 10. Et de même, au moment où e_5 a lieu, $\mathsf{DOA}(\{b_1\},t_1,\theta(e_5)) = 11{,}3 - 8{,}2 + 3{,}2 - 0 = 6{,}3$.

Ces deux événements ont des arcs de lecture en entrée qui correspondent à l'arc activateur entre t_1 et p_2.

Comme dans le cas des TPN, chaque événement impose des contraintes sur sa propre date d'occurrence mais également sur celles des autres événements. Détaillons simplement celles qui sont générées directement par l'événement e_4 :

$$\theta(e_4) \geq \theta(e_3) \text{ et } \theta(e_4) - \theta(e_3) + \theta(e_2) - 0 \in [10{,}20] \text{ et } \theta(e_1) = +\infty \text{ et } \theta(e_8) = +\infty \text{ et } \theta(e_4) \leq \theta(e_5)$$
$$\text{ou}$$
$$\theta(e_4) = +\infty \text{ et } (\theta(e_8) - \theta(e_3) + \theta(e_2) - 0 \leq 20 \text{ ou } \theta(e_5) - \theta(e_3) + \theta(e_2) - 0 \leq 20)$$

À partir de ces définitions, presque toute la théorie présentée pour les TPN s'applique et l'on peut prouver les mêmes résultats de façon très similaire : il s'agit, en simplifiant un peu, de remplacer $\theta(e) - \mathsf{TOE}(B,l(e),e)$ partout par $\mathsf{DOA}(E(B),l(e),\theta(e))$.

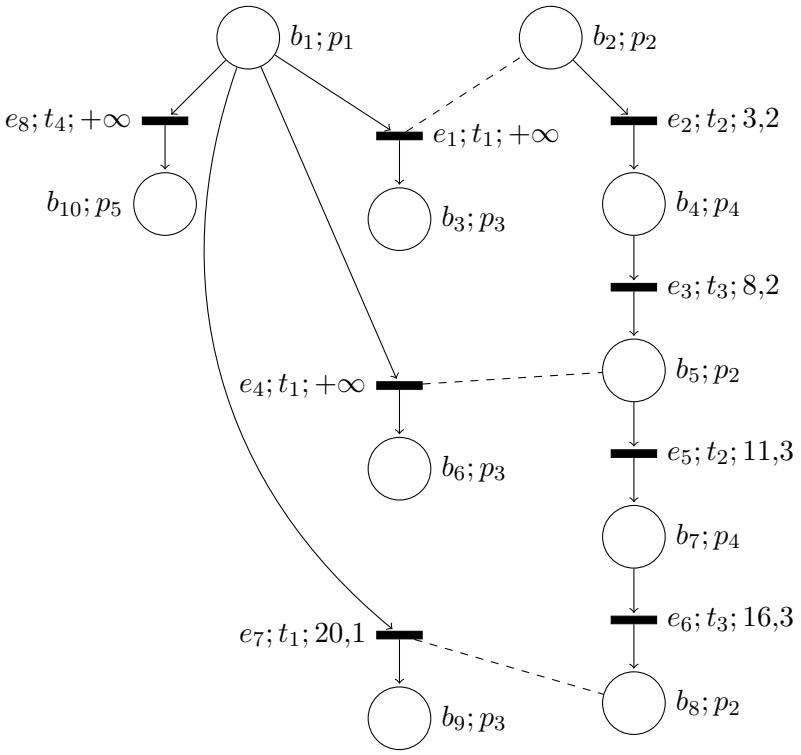


FIGURE 4.7 – Un processus de branchement temporel valide du réseau de la figure 4.6

Les seuls résultats que l'on ne peut pas retrouver sont bien sûr la proposition 3 et le théorème 10. En effet, à cause des contraintes sur des sommes de variables induites par DOA, l'ensemble des fonctions de temporisation valides pour un processus de branchement fini forment une union finie de polyèdres généraux et non de zones. Par conséquent, le caractère fini du préfixe sans événement d'arrêt du dépliage symbolique n'est plus assuré. Ces résultats sont cohérents avec ceux que l'on connaît pour les sémantiques séquentielles et notamment l'indécidabilité de l'accessibilité de marquage pour les SwPN [J5].

4.5.2 Paramètres

Nous nous intéressons maintenant aux réseaux de Petri temporels paramétrés [C10, J2]. Il s'agit de remplacer certaines bornes des intervalles par des paramètres ou des expressions linéaires sur ces paramètres. L'intérêt est de pouvoir modéliser un système pour lequel toutes les informations de temps ne sont pas connues puis de trouver des contraintes sur les paramètres assurant certaines propriétés.

Soit $\mathbb{Z}$ l'ensemble des entiers relatifs. Soit V un ensemble fini de variables appelées *paramètres*. Une *expression linéaire* (paramétrée) sur V est une expression engendrée par la grammaire :

$$\phi ::= k|x|k * \phi|\phi + \phi$$

où $k \in \mathbb{Z}$ et $x \in V$.

Un intervalle paramétré sur V est un intervalle dont la borne inférieure est soit $-\infty$ soit une expression linéaire et la borne supérieure, soit $+\infty$, soit une expression linéaire. On note $\mathcal{I}_p(V)$ l'ensemble des intervalles paramétrés sur V.

Une valuation ν sur V est une fonction de V dans $\mathbb{R}$. On note $\mathbb{R}^V$ l'ensemble des valuations sur V.

Pour une valuation ν et une expression linéaire e on note $\nu(e)$ le réel obtenu en remplaçant dans e chaque variable x par $\nu(x)$. De même, pour un intervalle paramétré $I = (e_1,e_2)$ on note $\nu(I)$ l'intervalle obtenu en appliquant ν à chacune des ses bornes (différentes de ∞).

Définition 51 (Réseau de Petri temporel paramétré)**.** Un *réseau de Petri temporel paramétré* sauf et avec arcs de lectures (PTPN pour *Parametric Time Petri Net*) est un sextuplet $\mathcal{N} = (P,T,F,F_r,m_0,V,I_s,V_0)$ tel que :

- (P,T,F,F_r,m_0) est un réseau de Petri qu'on note $\mathsf{Untimed}(\mathcal{N})$;
- V est un ensemble fini de *paramètres* ;
- $I_s : T \to \mathcal{I}_p(V)$ associe à chaque transition une *contrainte temporelle paramétrée* sous la forme d'un intervalle paramétré appelé *intervalle de tir statique* de la transition ;
- $V_0 \subseteq \mathbb{R}^V$ est un ensemble de valuations tel que pour toute transition $t \in T$ et toute valuation $\nu \in V_0$, $\nu(I_s(t)) \neq \emptyset$ et $\nu(I_s(t)) \subseteq \mathbb{R}_{\geq 0}$. On dit que ν est *valide* pour $\mathcal{N}$.

Pour toute valuation ν de V_0, on note $\nu(\mathcal{N})$ le réseau de Petri temporel obtenu en remplaçant chaque intervalle paramétré I par $\nu(I)$. On a bien un TPN grâce à la définition de V_0.

Exemple 19. La figure 4.8 présente une version paramétrée du réseau de la figure 4.4. On a ajouté deux paramètres, a et b (c'est-à-dire que $V = \{a,b\}$) ainsi qu'une contrainte initiale $V_0 = \{\nu \in \mathbb{R}^V | \nu(a) \geq 0 \text{ et } \nu(b) \geq 0\}$ (notée simplement $a \geq 0, b \geq 0$ sur la figure) qui assure que l'intervalle $[a,a+b]$ n'est pas vide et inclus dans $\mathbb{R}_{\geq 0}$.

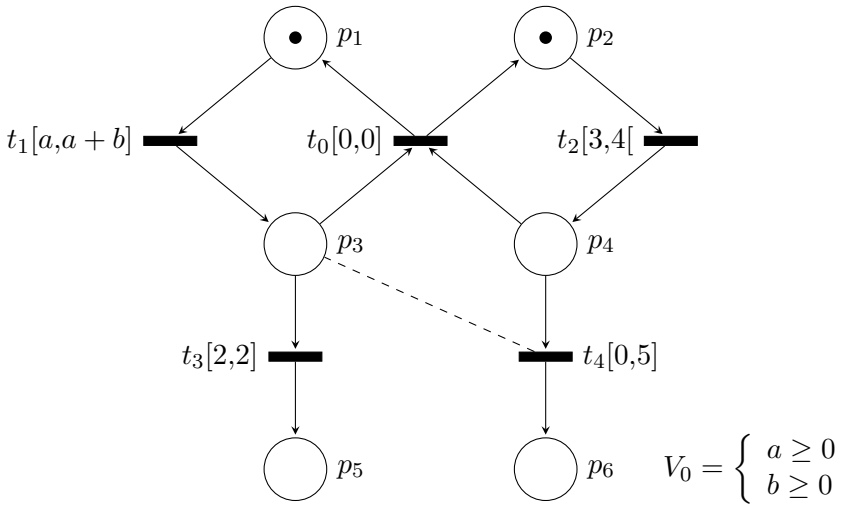


FIGURE 4.8 – Un réseau de Petri temporel paramétré

Comme précédemment, on définit une sémantique concurrente pour les PTPN sous la forme de processus temporels.

Définition 52 (Processus temporel d'un PTPN)**.** Un *processus temporel* d'un PTPN $\mathcal{N}$ est un triplet (C,θ,ν) tel que :

– ν est une valuation valide pour $\mathcal{N}$;
– (C,θ) est un processus temporel de $\nu(\mathcal{N})$.

Définition 53 (Processus temporel valide d'un PTPN)**.** Un processus temporel (C,θ,ν) d'un PTPN $\mathcal{N}$ est *valide* si (C,θ) est un processus temporel valide de $\nu(\mathcal{N})$.

Exemple 20. Associé à toute valuation ν telle que $\nu(a) \leq 1{,}52$, le processus de branchement temporel de la figure 4.5a est un processus de branchement du réseau de la figure 4.8.

Il en est de même pour le processus temporel de la figure 4.5b et toute valuation ν telle que $\nu(a) \leq 0{,}13$.

Nous pouvons alors définir de la même façon les processus de branchement temporels comme des triplets $(\mathcal{B},\theta,\nu)$ et on retrouve tous les résultats présentés pour les TPNs, en remplaçant partout $I_s(t)$ par $\nu(I_s(t))$, à l'exception, comme pour les SwPN, de la proposition 3 et du théorème 10.

En effet, un processus de branchement temporel symbolique contient l'union de tous les processus de branchement temporels valides partageant le même processus de branchement sous-jacent. Pour le décrire, il y a donc à la fois des contraintes sur les dates d'occurrence et des contraintes sur les paramètres, qui ne peuvent pas être représentées par des zones. Comme pour les SwPN, il faut avoir recours à des unions de polyèdres généraux. Comme précédemment, le caractère fini du préfixe n'est alors plus garanti, ce qui est cohérent avec les résultats d'indécidabilité connus sur ce modèle [J2].

Exemple 21. L'ensemble des couples (θ,ν) qui forment un processus de branchement temporel valide avec le processus de branchement de la figure 4.2 sont décrits par :

$$\begin{cases} \nu(a) \leq 0 \\ \nu(b) \geq 0 \\ \nu(a) \leq \theta(e_1) \leq \nu(a) + \nu(b) \\ 1 \leq \theta(e_1) \leq \nu(a) + \nu(b) \\ 3 \leq \theta(e_2) < 4 \\ \theta(e_3) = +\infty \\ \theta(e_4) = \max\{\theta(e_1),\theta(e_2)\} \\ \theta(e_5) = +\infty \end{cases} \text{ ou } \begin{cases} \nu(a) < 2 \\ \nu(b) \geq 0 \\ \theta(e_1) \leq \nu(a) + \nu(b) \\ \nu(a) \leq \theta(e_1) < 2 \\ 3 \leq \theta(e_2) < 4 \\ \theta(e_3) = \theta(e_1) + 2 \\ \theta(e_3) \leq \theta(e_2) \\ \theta(e_4) = +\infty \\ \theta(e_5) = +\infty \end{cases} \text{ ou } \begin{cases} \nu(a) < 2 \\ \nu(b) \geq 0 \\ \theta(e_1) \leq \nu(a) + \nu(b) \\ \nu(a) \leq \theta(e_1) < 2 \\ 3 \leq \theta(e_2) < 4 \\ \theta(e_3) = \theta(e_1) + 2 \\ \theta(e_3) = \theta(e_2) \\ \theta(e_4) = +\infty \\ \theta(e_5) = \theta(e_2) \end{cases}$$

4.6 Supervision

Dans le cadre de réseaux paramétrés ou à chronomètres, l'existence d'un préfixe fini du dépliage symbolique n'est pas assurée. Nous présentons ici une technique de supervision, issue de [44], et qui permet de se passer de ce préfixe en utilisant des observations pour contraindre le dépliage.

Nous considérons désormais des réseaux de Petri temporels dont les transitions sont *étiquetées* par une fonction $\lambda : T \to \Sigma \cup \{\epsilon\}$, où Σ est un alphabet fini et $\epsilon \notin \Sigma$ représente à la fois le mot vide et l'action inobservable.

L'objectif est d'observer l'exécution d'un réseau de Petri temporel (éventuellement étendu avec des paramètres ou des chronomètres) à travers ces étiquettes et d'en déduire, à l'aide du dépliage, les comportements complets qui ont pu conduire à une observation donnée. On considère ici un type simple d'observation qui « compte » les différents type d'événements :

Définition 54 (Observation)**.** Une *observation* $\mathcal{O} : \Sigma \to \mathbb{N}$ est un multi-ensemble sur Σ.

On pourrait aussi introduire de la causalité ou de la concurrence explicite dans l'observation sans difficulté [C7].

En notant $\Sigma = \{a_1, \ldots, a_n\}$, pour toute observation $\mathcal{O}$, on définit son *vecteur de Parikh* $\varpi(\mathcal{O}) = (\mathcal{O}(a_1), \ldots, \mathcal{O}(a_n))$. De façon similaire, pour une configuration E, on définit son *vecteur de Parikh* $\varpi(E) = (|\lambda^{-1}(a_i)|)_{i \in [1..n]}$, en étendant λ à E par $\forall e \in E, \lambda(e) = \lambda(l(e))$. Pour deux vecteurs de Parikh ϖ_1 et ϖ_2, on note $\varpi_1 = \varpi_2$ si pour tout i, la i^{e} composante de ϖ_1 est égale à la i^{e} composante de ϖ_2.

On peut alors définir quelles configurations sont compatibles avec une observation donnée puis étendre cette notion aux processus temporels et finalement aux processus de branchement temporels.

Définition 55 (Compatibilité)**.** Une configuration E est compatible avec une observation $\mathcal{O}$ si $\varpi(E) = \varpi(\mathcal{O})$.

Un processus temporel (E,θ) est compatible avec $\mathcal{O}$ si E l'est.

Un processus de branchement temporel $\mathcal{B} = (B,E,F,F_r,m_0,\theta)$ est compatible avec $\mathcal{O}$ s'il est temporellement complet et (E,θ) est compatible avec $\mathcal{O}$.

On peut ainsi construire un *dépliage symbolique compatible* avec $\mathcal{O}$ qui est l'union des processus de branchement temporels compatibles et utiliser le théorème 6 pour extraire les processus temporels correspondants : ces processus correspondent aux « explications » de l'observation et, comme précédemment, on peut prendre l'union des processus partageant une même configuration pour obtenir des explications « symboliques », c'est-à-dire pour chacune d'elle un ensemble structuré par la causalité d'événements ayant pu avoir lieu dans la même histoire et ayant produit l'observation, ainsi que les contraintes sur les dates de ces événements.

En l'absence de boucles d'événements inobservables, le nombre de ces explications symboliques est fini :

Théorème 11 ([C7])**.** *Si le réseau ne contient pas de boucles de transitions étiquetées par ϵ, alors, pour toute observation $\mathcal{O}$, le dépliage symbolique compatible avec $\mathcal{O}$ est fini.*

Si de telles boucles existent, on peut borner *a priori* le nombre d'événements inobservables et ne pas déplier le réseau selon les branches qui contiennent plus que ce nombre fixé d'événements inobservables. Évidemment, en général, le risque est de ne pas obtenir certaines explications valides de l'observation. D'autres stratégies sont étudiées, dans le cas non temporisé, dans [48].

Exemple 22. La figure 4.9 présente un réseau de Petri temporel paramétré étiqueté. Ici $\Sigma = \{t\}$ et la figure montre directement les étiquettes au lieu des noms des transitions. Un préfixe du dépliage symbolique est explicité sur la figure 4.10.

Enfin, deux explications symboliques de l'observation $\{t,t\}$ sont données sur les figures 4.11a et 4.11b.

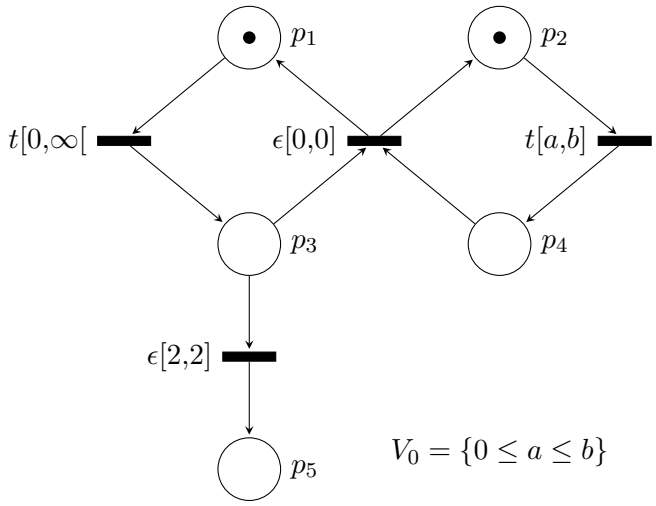


FIGURE 4.9 – Un PTPN

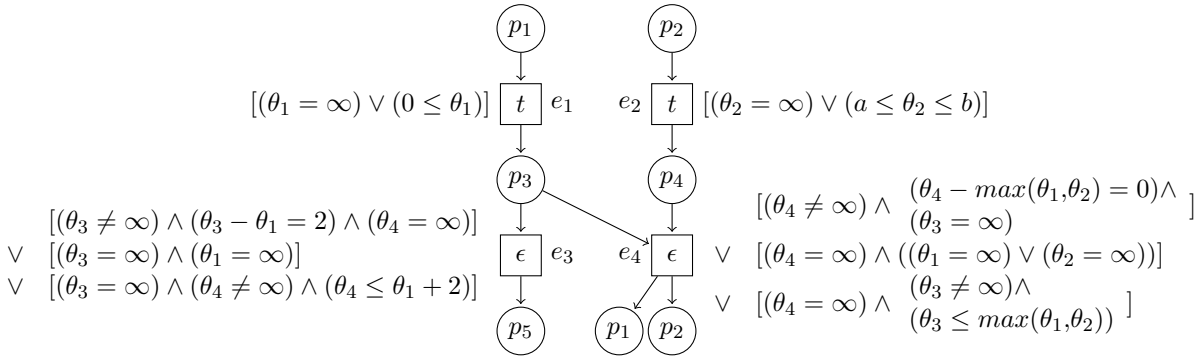


FIGURE 4.10 – Un préfixe du dépliage symbolique du PTPN de la figure 4.9.

4.7 Conclusion

Nous avons présenté une nouvelle approche pour le dépliage symbolique de réseaux de Petri temporels (saufs). Celle-ci est basée sur la notion de conflit direct. Nous avons montré comment calculer un préfixe complet et fini de ce dépliage symbolique.

Nous avons également montré comment étendre l'approche pour les réseaux à chronomètres ou paramétrés. Dans ces deux cas, l'existence d'un préfixe complet est toujours garantie mais son caractère fini ne l'est plus. Cela n'interdit cependant pas toute tentative d'analyse de ces modèles : l'expérience montre que même pour des modèles très expressifs tels que les automates hybrides linéaires par exemple [49], malgré l'indécidabilité des problèmes intéressants tels que l'accessibilité, des succès pratiques peuvent êtres obtenus.

De plus, nous pouvons également appliquer un certain nombre de techniques qui n'explorent qu'un sous-ensemble fini du dépliage symbolique : nous avons évoqué ici la technique de supervision Fabre et co-auteurs [44]. L'utilisation de cette technique dans ce cadre est illustrée de façon plus complète par une étude de cas dans [C7].

Nous considérons plusieurs perspectives d'extension de ces travaux dans le proche avenir. Dans un premier temps, il semble raisonnablement facile d'étendre notre approche aux

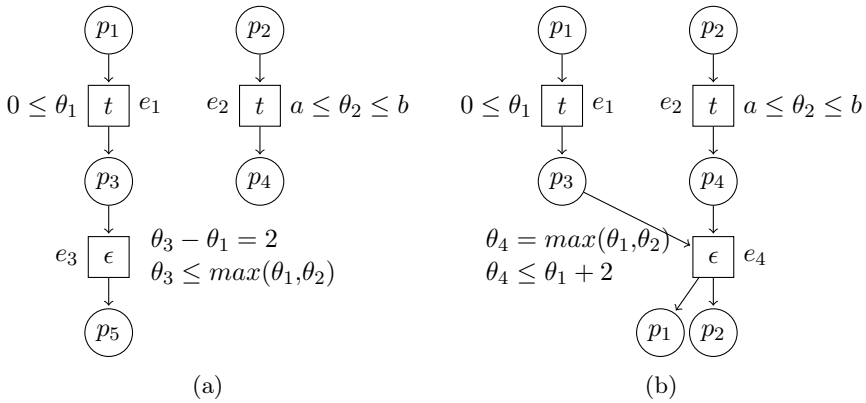


FIGURE 4.11 – Deux explications pour l'observation $\{t,t\}$ pour le PTPN de la figure 4.9.

réseaux d'automates temporisés. Il serait également intéressant de trouver une notion plus locale que Earlier pour la définition du préfixe complet. Nous souhaitons également réaliser une implémentation efficace de cette approche.

Bibliographie

[1] K. Altisen and S. Tripakis. Tools for controller synthesis of timed systems. In *Proc. 2nd Work. on Real-Time Tools (RT-TOOLS'02)*, 2002. Proc. published as Technical Report 2002-025, Uppsala University, Sweden.

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

[3] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing*, pages 592–601, 1993.

[5] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller Synthesis for Timed Automata. In *Proc. IFAC Symp. on System Structure & Control*, pages 469–474. Elsevier Science, 1998.

[6] T. Aura and J. Lilius. A causal semantics for time Petri nets. *Theoretical Computer Science*, 243(2):409–447, 2000.

[7] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[8] P. Baldan, A. Corradini, B. König, and S. Schwoon. McMillan's complete prefix for contextual nets. *Transactions on Petri Nets and Other Models of Concurrency*, 1:199–220, Nov. 2008. Volume 5100 of Lecture Notes in Computer Science.

[9] G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, June 2006.

[10] G. Benattar. *Synthèse de systèmes informatiques non-interférents*. PhD thesis, Université de Nantes, Nantes, France, 2011.

[11] 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, March 1991.

[12] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Information Processing: proceedings of the IFIP congress 1983*, volume 9 of *IFIP congress series*, pages 41–46. Elsevier Science Publishers, Amsterdam, 1983.

[13] P. Bouyer. Untameable timed automata ! In H. Alt and M. Habib, editors, *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631, Berlin, Germany, Feb. 2003. Springer.

[14] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In W. A. Hunt, Jr and F. Somenzi, editors, *15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192, Boulder, Colorado, USA, July 2003. Springer.

[15] P. Bouyer, S. Haddad, and P.-A. Reynier. Timed unfoldings for networks of timed automata. In S. Graf and W. Zhang, editors, *Proceedings of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 292–306, Beijing, China, Oct. 2006. Springer.

[16] M. Boyer and O. H. Roux. On the compared expressiveness of arc, place and transition time Petri nets. *Fundamenta Informaticae*, 88(3):225–249, 2008.

[17] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: a Model-Checking Tool for Real-Time Systems. In *Proc. 10th Conf. on Computer Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 546–550. Springer, 1998.

[18] L. Bozzelli and S. L. Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009.

[19] 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.

[20] G. Bucci, R. Piovosi, L. Sassoli, and E. Vicario. Introducing probability within state class analysis of dense-time-dependent systems. In *2nd International Conference On the Quantitative Evaluation of Systems (QEST'05)*, pages 13 – 22, Sept. 2005.

[21] P. Bulychev, T. Chatain, A. David, and K. G. Larsen. Efficient on-the-fly algorithm for checking alternating timed simulation. In *Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems*, number 5813 in LNCS, pages 73–87. Springer, 2009.

[22] J. Burch, E. Clarke, and K. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation (Issue for LICS90 best papers)*, 98(2):153–181, 1992.

[23] F. Cassez, T. Chatain, and C. Jard. Symbolic Unfoldings for Networks of Timed Automata. In S. Graf and W. Zhang, editors, *Proceedings of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 307–321, Beijing, China, Oct. 2006. Springer.

[24] F. Cassez, J. Jessen, K. Larsen, J.-F. Raskin, and P.-A. Reynier. Automatic Synthesis of Robust and Optimal Controllers – An Industrial Case Study. In *Proc. of the 12th International Conference on Hybrid Systems: Computation and Control (HSCC'09)*, number 5469 in Lecture Notes in Computer Science, San Francisco, CA, USA, Apr. 2009. Springer.

[25] F. Cassez and K. Larsen. The impressive power of stopwatches. In C. Palamidesi, editor, *11th International Conference on Concurrency Theory, (CONCUR'2000)*, number 1877 in LNCS, pages 138–152, University Park, P.A., USA, July 2000. Springer-Verlag.

[26] F. Cassez, J. Mullins, and O. H. Roux. Synthesis of non-interferent systems. In *4th Int. Conf. on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS'07)*, volume 1 of *Communications in Computer and Information Science*, pages 307–321. Springer, 2007.

[27] T. Chatain and C. Jard. Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In *Proceedings of ICATPN*, volume 4024 of *LNCS*, pages 125–145. Springer, 2006.

[28] T. Chatain and V. Khomenko. On the well-foundedness of adequate orders used for construction of complete unfolding prefixes. *Information Processing Letters*, 104(4):129–136, 2007.

[29] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[30] J.-M. Couvreur and Y. Thierry-Mieg. Hierarchical Decision Diagrams to Exploit Model Structure. In *Formal Techniques for Networked and Distributed Systems (FORTE'05)*, volume 3731 of *LNCS*, pages 443–457. Springer, 2005.

[31] A. David. The UPPAAL DBM library: `http://people.cs.aau.dk/~adavid/UDBM/`, 2012.

[32] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. ECDAR: An environment for compositional design and analysis of real time systems. In *Proceedings of Automated Technology for Verification and Analysis*, volume 6252 of *LNCS*, pages 365–370. Springer, 2010.

[33] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Methodologies for specification of real-time systems using timed I/O automata. In *Formal Methods for Components and Objects*, volume 6286 of *LNCS*, pages 290–310. Springer, 2010.

[34] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed I/O automata: a complete specification theory for real-time systems. In *Proc. of the 13th International Conference on Hybrid Systems: Computation and Control (HSCC'10)*, pages 91–100. ACM, 2010.

[35] A. David, K. G. Larsen, S. Li, and B. Nielsen. Cooperative testing of uncontrollable real-time systems. In *4th workshop of Model-Based Testing (MBT'08)*, 2008.

[36] A. David, K. G. Larsen, S. Li, and B. Nielsen. A game-theoretical approach to real-time system testing. In *Proceedings of the 11th International Conference on Design Automation and Test in Europe (DATE'08)*, 2008.

[37] A. David, K. G. Larsen, S. Li, and B. Nielsen. Timed testing under partial observability. In *2nd IEEE International Conference on Software Testing, Verification, and Validation (ICST'09)*, 2009.

[38] L. de Alfaro, T. A. Henzinger, and O. Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386:188–217, October 2007.

[39] L. de Alfaro, T. A. Henzinger, and R. Majumdar. Symbolic Algorithms for Infinite-State Games. In *Proc. 12th Conf. on Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 536–550. Springer, 2001.

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

[41] J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23:151–195, 1994.

[42] J. Esparza and K. Heljanko. *Unfoldings, A Partial-Order Approach to Model Checking.* Monographs in Theoretical Computer Science. Springer, 2008.

[43] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

[44] E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. *Discrete Event Dynamic Systems*, 15(1):33–84, 2005.

[45] R. Focardi and R. Gorrieri. Classification of security properties (part I: Information flow). In *Foundations of Security Analysis and Design I: FOSAD 2000 Tutorial Lectures*, volume 2171 of *Lecture Notes in Computer Science*, pages 331–396. Springer-Verlag, 2001.

[46] G. Gardey, J. Mullins, and O. H. Roux. Non-interference control synthesis for security timed automata. *Electronic Notes in Computer Science*, 180(1):35–53, Aug. 2005. 3rd International Workshop on Security Issues in Concurrency (SecCo'05).

[47] 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*, 6(3):301–320, 2006.

[48] B. Grabiec. *Supervision of distributed systems using constrained unfoldings of timed models.* PhD thesis, École Normale Supérieure de Cachan, Rennes, France, 2011.

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

[50] 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.

[51] G. J. Holzmann. *The SPIN Model Checker.* Addison-Wesley, 2003.

[52] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

[53] J. J. Jessen, J. I. Rasmussen, K. G. Larsen, and A. David. Guided controller synthesis for climate controller using UPPAAL-TIGA. In *Proceedings of the 19th International Conference on Formal Modeling and Analysis of Timed Systems*, number 4763 in LNCS, pages 227–240. Springer, 2007.

[54] N. Jones, L. Landweber, and Y. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

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

[56] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1-2):134–152, 1997.

[57] X. Liu and S. Smolka. Simple Linear-Time Algorithm for Minimal Fixed Points. In *Proc. 26th Conf. on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *LNCS*, pages 53–66. Springer, 1998.

[58] O. Maler, A. Pnueli, and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems. In *Proc. 12th Symp. on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *LNCS*, pages 229–242. Springer, 1995.

[59] K. L. McMillan. Using unfolding to avoid the state space explosion problem in the verification of asynchronous circuits. In *Proceedings of Computer Aided Verification (CAV'92)*, volume 663 of *LNCS*, pages 164–177. Springer, 1992.

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

[61] A. Pnueli. The temporal logic of programs. *18th Annual IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[62] P. Ramadge and W. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77:81–98, 1989.

[63] W. Thomas. On the Synthesis of Strategies in Infinite Games. In *Proc. 12th Symp. on Theoretical Aspects of Computer Science (STACS'95)*, volume 900, pages 1–13. Springer, 1995. Invited talk.

[64] L.-M. Traonouez. *Vérification et dépliages de réseaux de Petri temporels paramétrés.* Ph.D. thesis, Université de Nantes, Nantes, France, Nov. 2009.

[65] S. Tripakis and K. Altisen. Controller synthesis for discrete and dense-time systems. In *Proc. World Congress on Formal Methods in the Development of Computing Systems (FM'99)*, volume 1708 of *LNCS*, pages 233–252. Springer, 1999.

[66] W. Vogler, A. L. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In D. Sangiorgi and R. de Simone, editors, *9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 501–516, Nice, France, Sept. 1998. Springer.

Annexe A

Preuves pour le chapitre 3

Proposition 1. *Si f est une stratégie gagnante pour $(\mathcal{G}',W')$ alors sa restriction $f_{\mathcal{G}}$ à $\mathsf{Runs}(\mathcal{G})$ est une stratégie gagnante pour $(\mathcal{G},W)$.*

Démonstration. Supposons que f soit une stratégie gagnante pour le jeu $(\mathcal{G}',W')$. Soit q_0 l'état initial de (la sémantique) de $\mathcal{G}'$ (et de $\mathcal{G}$). On sait que puisque f est gagnante, $\mathsf{MaxOut}(\mathcal{G}',q_0,f) \subseteq W'$.

Soit ρ une exécution de $\mathsf{MaxOut}(\mathcal{G},q_0,f_{\mathcal{G}})$. On prouve que $\rho \in W$.

On a de façon évidente $\rho \in \mathsf{Outcome}(\mathcal{G}',q_0,f)$. Et puisque $\rho \in \mathsf{MaxOut}(\mathcal{G},q_0,f_{\mathcal{G}})$ on a :

- soit ρ a un nombre infini d'actions discrètes, ou ρ a un nombre fini d'actions discrètes mais sa durée est infinie. Alors $\rho \in \mathsf{MaxOut}(\mathcal{G}',q_0,f)$ et donc $\rho \in W'$. ρ ne peut pas être dans R' car $R' \cap \mathsf{Runs}(\mathcal{G}) = \emptyset$. Donc $\rho \in W$.
- soit ρ a un nombre fini d'actions discrètes et une durée finie. Alors, dans $\mathcal{G}$, le temps est bloqué dans $\mathsf{Last}(\rho)$ à cause d'un invariant et comme ρ est maximal dans $\mathcal{G}$, il n'existe pas d'action discrète qui puisse étendre ρ en une autre exécution de $\mathsf{Outcome}(\mathcal{G}',q_0,f)$. Donc, dans $\mathcal{G}'$, ρ peut être étendue par un écoulement infini du temps en une exécution maximale ρ'. Puisque f est gagnante pour $(\mathcal{G}',W')$ alors $\rho' \in W'$. Il n'y a pas d'action du joueur 2 qui puisse être jouée à partir de $\mathsf{Last}(\rho)$.
  Si le temps est bloqué à cause d'un invariant non-strict alors pour tout $d \in \mathbb{R}_{>0}$, et tout q tel que $\rho \xrightarrow{d}_{\mathcal{G}'} q$, $q \notin \mathsf{States}(\mathsf{Runs}(\mathcal{G}))$ et donc aucune action du joueur 2 n'est possible depuis q.
  Si le temps est bloqué à cause d'un invariant strict alors, si ρ'' est obtenu par écoulement du temps dans $\mathcal{G}'$ à partir de ρ, soit $\rho'' \in \mathsf{Runs}(\mathcal{G})$ et, par définition de η, il n'y a pas de nouvelle action du joueur 2 jouable après ρ'', soit si, on a écoulé suffisamment de temps, $\rho'' \notin \mathsf{Runs}(\mathcal{G})$ et donc aucune action du joueur 2 n'est jouable.
  Dans ces deux cas, on a donc $\rho' \notin R'$ et donc $\rho \in W$.

□

Proposition 2. *Soit f une stratégie gagnante pour $(\mathcal{G},W)$. Soit f' la stratégie définie par $f'(\rho) = \mathsf{delay}$ si $f(\rho) = \bot$ ou $\rho \notin \mathsf{Runs}(\mathcal{G})$ et $f'(\rho) = f(\rho)$ sinon. Alors f' est une stratégie gagnante pour $(\mathcal{G}',W')$.*

Démonstration. Supposons que f soit une stratégie gagnante pour $(\mathcal{G},W)$. Alors, si q_0 est l'état initial, $\mathsf{MaxOut}(\mathcal{G},q_0,f) \subseteq W$. f' est trivialement bien définie. Soit $\rho' \in \mathsf{MaxOut}(\mathcal{G},q_0,f')$. On prouve que $\rho' \in W'$.

Si $\rho' \in \mathsf{Runs}(\mathcal{G})$ alors on a de façon immédiate $\rho' \in \mathsf{MaxOut}(\mathcal{G},q_0,f)$. Et comme f est gagnante, $\rho' \in W$ et donc $\rho' \in W'$.

Si $\rho' \notin \mathsf{Runs}(\mathcal{G})$ alors ρ' a une nombre fini d'actions discrètes et ρ' est l'extension par un écoulement infini du temps d'une exécution ρ de $\mathsf{Outcome}(\mathcal{G},q_0,f)$. Si ρ est maximale dans $\mathsf{Outcome}(\mathcal{G},q_0,f)$, alors puisque f est gagnante, et par définition de W', on a $\rho' \in W'$. Sinon, on peut considérer sans perte de généralité que :

- soit l'invariant de la localité de $\mathsf{Last}(\rho)$ est non strict et alors plus aucun écoulement de temps n'est possible depuis ρ dans $\mathsf{Outcome}(\mathcal{G},q_0,f)$. Alors on a nécessairement $f(\rho) \neq \mathsf{delay}$. De plus, aucune action du joueur 1 n'est non plus possible depuis $\mathsf{Last}(\rho)$, sinon on aurait eu $f(\rho) \in \Sigma_1$ et ρ' n'aurait pas été dans $\mathsf{Outcome}(\mathcal{G}',q_0,f')$. Donc, puisque ρ n'est pas maximal dans $\mathsf{Outcome}(\mathcal{G},q_0,f)$, la seule possibilité est qu'il existe $a \in \Sigma_2$ tel que $\rho \xrightarrow{a}$, et donc $\rho \in R' \subseteq W'$.
- soit l'invariant de la localité de $\mathsf{Last}(\rho)$ est strict et moins de η unités de temps peuvent s'écouler. S'il existait une action du joueur 1 jouable à partir de $\mathsf{Last}(\rho)$, alors d'après la définition 14, il existerait une extension ρ'' de ρ par écoulement du temps telle que $f(\rho'') \neq \mathsf{delay}$. Comme précédemment, ρ' ne serait pas dans $\mathsf{Outcome}(\mathcal{G}',q_0,f')$. Donc, à nouveau, comme ρ n'est pas maximale dans $\mathsf{Outcome}(\mathcal{G},q_0,f)$, la seule possibilité est qu'il existe une action $a \in \Sigma_2$ telle que $\rho \xrightarrow{a}$. Par définition de η, on sait alors que a est toujours possible pour tes les extensions par délai de ρ dans $\mathsf{Outcome}(\mathcal{G},q_0,f)$ et donc $\rho \in R' \subseteq W'$.

□

Lemme 2. *Si F est croissante, lorsqu'on l'exécute sur TGS, la boucle while de l'algorithme SOTFBLF a les propriétés d'invariance suivantes :*

I1 *pour tout* $S \in \mathrm{Passed}$, *si* $S \xrightarrow{e} S'$ *alors*
- *soit* $(S,e,S') \in \mathrm{Waiting}$;
- *soit* $S' \in \mathrm{Passed}$ *et* $(S,e,S') \in \mathrm{Depend}[S']$;

I2 *si* $q \in \mathrm{Win}[S]$ *pour un* $S \in \mathrm{Passed}$ *alors* $q \in \mu X.F(X)$;

I3 *si* $q \in S \setminus \mathrm{Win}[S]$ *pour un* $S \in \mathrm{Passed}$ *alors*
- *soit il existe* $(S,e,S') \in \mathrm{Waiting}$ *avec* $S' \in \mathrm{Passed}$;
- *soit* $q \notin F(\mathrm{Win}[S] \cup \bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(S,e')])$.

Démonstration. Considérons l'invariant **I1**. Lorsque l'on entre dans la boucle *while* l'invariant est vrai car $(S_0,e,S') \in \mathrm{Waiting}$ pour tous les successeurs symboliques de S_0. Supposons l'invariant vrai après l'itération n et soit (S,e,S') la transition (symbolique) considérée pendant l'itération $n+1$. Si $S' \notin \mathrm{Passed}$ avant cette itération, alors maintenant $S' \in \mathrm{Passed}$ et $(S,e,S') \in \mathrm{Depend}[S']$. De plus, tous les successeurs de S' sont ajoutés à Waiting assurant que l'invariant est toujours vrai après cette itération. Si $S' \in \mathrm{Passed}$ avant l'itération $n+1$ alors $(S,e,S') \in \mathrm{Depend}[S']$ grâce à l'instruction ligne 23.

Considérons l'invariant **I2**, c'est-à-dire, pour tout $S \in \mathrm{Passed}$, $\mathrm{Win}[S] \subseteq \mu X.F(X)$. Lorqu'on entre dans la boucle *while* cette propriété est trivialement vraie. Supposons la propriété vraie après l'itération n et considérons l'itération $n+1$, sur la transition (S,e,S'). Si l'on est alors dans l'exploration en avant, $\mathrm{Win}[S]$ est modifié ligne 12 et après cette instruction la propriété est trivialement vraie. Sinon, $\mathrm{Win}[S]$ est modifié à la ligne 21 lors de la propagation en arrière. Par l'hypothèse d'induction, on sait que $\mathrm{Win}[S] \subseteq \mu X.F(X)$ et, pour

chacun des successeurs S'' de S, $\mathrm{Win}[S''] \subseteq \mu X.F(X)$ si $S'' \in$ Passed et $\mathrm{Win}[S''] = \emptyset$ sinon. Donc $\mathrm{Win}[S] \cup \bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(S,e)] \subseteq \mu X.F(X)$. Et puisque F est croissante, on a $\mathrm{Win}^* \subseteq \mu X.F(X)$. Et donc la propriété et vraie après la ligne 21.

Considérons l'invariant **I3**. À l'entrée dans la boucle, la propriété est vraie car seul S_0 est dans Passed et donc $\bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(S_0,e')] = \emptyset$. On a donc $\mathrm{Win}[S_0] = S_0 \cap F(\emptyset)$, et donc bien si $q \notin \mathrm{Win}[S_0]$ alors $q \notin F(\emptyset)$.

Supposons maintenant la propriété vraie après l'itération n et considérons l'itération $n+1$. Soit (S,e,S') la transition traitée par cette itération.

Soit $T \in$ Passed et soit $q \in T \setminus \mathrm{Win}[T]$.

Supposons que l'itération $n+1$ se fasse en avant : $S' \notin$ Passed. Si S' n'est pas un successeur de T alors $\mathrm{Win}[T]$, ainsi que $F(\mathrm{Win}[T] \cup \bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(T,e')])$ sont inchangés par l'itération. Si S' est un successeur de T alors soit $\mathrm{Win}[S'] \neq \emptyset$, changeant potentiellement $F(\mathrm{Win}[T] \cup \bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(T,e')])$ mais alors les lignes 14–15 et la propriété **I1** assurent qu'une transition (T,e'',S') sera dans Waiting, soit $\mathrm{Win}[S']$ est inchangé et donc $F(\mathrm{Win}[T] \cup \bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(T,e')])$ également. Dans les deux cas, s'il existait $(T,e',T') \in$ Waiting avec $T' \in$ Passed alors c'est toujours le cas, car $S' \notin$ Passed. Et donc la propriété est toujours vérifiée.

Supposons mainteant qu'au contraire l'itération se fasse en arrière : $S' \in$ Passed. Si $T \neq S$ et S n'est pas un successeur de T alors la propriété est trivialement vraie après l'itération $n+1$ grâce à l'hypothèse d'induction. Si $T \neq S$ mais que $T \xrightarrow{e'} S$ alors la propriété **I1** assure que soit $(T,e',S) \in$ Waiting soit $(T,e',S) \in \mathrm{Depend}[S]$. Dans le premier cas, la propriété **I3** est bien satisfaite car (T,e',S) sera toujours dans Waiting après l'itération $n+1$. Dans le deuxième cas, les lignes 19 à 21 assurent que (T,e',S) sera bien dans Waiting après l'itération $n+1$ si $\mathrm{Win}[S]$ devient plus grand. Si $\mathrm{Win}[S]$ est inchangé par l'itération $n+1$, alors $F(\mathrm{Win}[T] \cup \bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(T,e')])$ également. Et donc, si $q \notin F(\mathrm{Win}[T] \cup \bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(T,e')])$ avant l'itération $n+1$, c'est toujours le cas après. S'il existait plutôt une transition $(T,e'',T') \in$ Waiting pour un $T' \in$ Passed, alors c'est toujours le cas car $T \neq S$. Supposons finalement que $T = S$, alors après l'itération $n+1$, soit $q \in \mathrm{Win}[T]$ et la propriété est vérifiée par défaut, soit $q \notin \mathrm{Win}[T]$ et trivialement, $q \notin F(\mathrm{Win}[T] \cup \bigcup_{e' \in E} \mathrm{Win}[\mathsf{Next}(T,e')])$. □

On peut maintenant énoncer et prouver le résultat de correction :

Théorème 1. *Si F vérifie les propriétés* **F1** *et* **F2**, *alors à la terminaison de l'exécution de l'algorithme SOTFBLF sur une TGS, les deux propriétés suivantes sont vraies :*

1. *si $q \in \mathrm{Win}[S]$ pour un $S \in$ Passed alors $q \in \mu X.F(X)$;*
2. *si $\mathrm{Waiting} = \emptyset$ et $q \in S \setminus \mathrm{Win}[S]$ pour un $S \in$ Passed alors $q \notin \mu X.F(X)$.*

Démonstration. La propriété **I2** du lemme 2 assure immédiatement que si $q \in \mathrm{Win}[S]$ pour un $S \in$ Passed alors $q \in \mu X.F(X)$.

Pour la deuxième propriété, considérons les ensembles $U = \{q \in Q | \exists S \in \text{Passed t.q. } q \in S \setminus \mathrm{Win}[S]\}$, Q étant l'ensemble des états accessibles de la TGS, et $V = Q \setminus U$.

On montre que $F(V) \subseteq V$ et donc $\mu X.F(X) \subseteq V$. Alors, si $q \in U$, $q \notin V$ et donc $q \notin \mu X.F(X) \subseteq V$.

Soit $q \in F(V)$. Alors d'après la contrainte **F2** sur F, il existe $V' \subseteq \{q\}^{\nearrow} \cup \bigcup_{e \in E} \mathsf{Succ}(\{q\}^{\nearrow}, e)$ tel que $V' \subseteq V$ et $q \in F(V')$. Et comme F est croissante, $q \in \{q\}^{\nearrow} \cup \bigcup_{e \in E} \mathsf{Succ}(\{q\}^{\nearrow}, e)$.

Si $S \in$ Passed est tel que $q \in S$ alors $V \cap \{q\}^{\nearrow} \cup \bigcup_{e \in E} \mathsf{Succ}(\{q\}^{\nearrow}, e) \subseteq V \cap S \cup \bigcup_{e \in E} \mathsf{Next}(S,e)$ et donc $q \in F((V \cap S) \cup \bigcup_{e \in E} V \cap \mathsf{Next}(S,e))$.

Supposons que $q \notin V$ alors $q \notin \mathrm{Win}[S]$. Donc d'après la propriété **I3** du lemme 2, puisque Waiting $= \emptyset$, $q \notin F(\mathrm{Win}[S] \cup \bigcup_{e \in E} \mathrm{Win}[\mathsf{Next}(S, e)])$. On alors $q \notin F((V \cap S) \cup \bigcup_{e \in E} V \cap \mathsf{Next}(S, e)])$. ce qui résulte en une contradiction. Donc $q \in V$. □

Théorème 2. *Si F vérifie les propriétés* **F1** *et* **F3** *alors la terminaison de l'algorithme SOTF-BLF est garantie.*

Démonstration. Si F vérifie **F3** alors pour tout S, $\mathrm{Win}[S]$ est un ensemble fini de régions. Par ailleurs, il existe un nombre fini d'états symboliques S. Enfin, chaque transition symbolique $\sigma = (S,e,S')$ sera présente dans la liste Waiting au plus $1 + |S']$ fois, si $|S'|$ est le nombre de régions dans S'. σ sera dans Waiting une première fois lorsque S sera calculé et à nouveau à chaque fois que $\mathrm{Win}[S']$ grossit. $\mathrm{Win}[S']$ ne peut que grossir car F est croissante, et à chaque fois d'au moins une région, et pas plus de $|S'|$ fois car $\mathrm{Win}[S'] \subseteq S$. □

Annexe B

Preuves pour le chapitre 4

Lemme 3. *Soit E' un ensemble d'événements. $\#E'$ ssi :*
- *$\exists e_1,e_2 \in \lceil E' \rceil$ t.q. $e_1 \,\mathsf{conf}\, e_2$;*
- *ou $\mathsf{RCycles}(\lceil E' \rceil) \neq \emptyset$.*

Démonstration. L'implication de la droite vers la gauche est triviale. Nous nous concentrons donc sur l'implication inverse. Supposons donc que $\#E'$. Si l'une des deux conditions de la définition 27 est satisfaite et que l'ensemble des préconditions B des événements correspondants n'est pas en conflit alors on a le résultat attendu. Supposons donc que B est en conflit. Alors nous avons un nouvel ensemble d'événements $E'' = {}^\bullet B$ tel que $\#E''$ et $E'' \subset E'$. Nous pouvons ainsi répéter notre raisonnement pour E'' mais puisque $\subset$ est, par définition des processus de branchements, bien fondée sur les ensembles d'événements, nous ne pourrons pas répéter ce raisonnement un nombre infini de fois, ce qui nous donne le résultat attendu. □

Lemme 4 ([6])**.** *Soit (C,θ) un processus temporel valide d'un TPN.*

$$\forall e,e' \in C, e < e' \Rightarrow \theta(e) \leq \theta(e')$$

Lemme 5. *Soit $(\mathcal{B},\theta)$ un TBP valide d'un TPN et E son ensemble d'événements.*

$$\forall e,e' \in E, e < e' \Rightarrow \theta(e) \leq \theta(e')$$

Démonstration. Comme $e < e'$, il existe un chemin entre e et e', composé des événements $e_1,e_2,\ldots,e_n$ avec $e_1 = e$ et $e_n = e'$. On a en outre pour tout $i \in [1..n]$, $e_i^\bullet \cap ({}^\bullet e_{i+1} \cup {}^\circ e_{i+1}) \neq \emptyset$. Donc comme $(\mathcal{B},\theta)$ est valide, si $\theta(e_{i+1}) \neq +\infty$ alors, par l'équation 4.4 et par définition de TOE, $\theta(e_i) \leq \theta(e_{i+1})$. Si $\theta(e_{i+1}) = +\infty$ alors de façon évidente on a également $\theta(e_i) \leq \theta(e_{i+1})$. Et donc par transitivité, $\theta(e) \leq \theta(e')$. □

Lemme 6. *Soit $(\mathcal{B},\theta)$ un TBP valide d'un TPN et E son ensemble d'événements. L'ensemble $E_{<+\infty}$ défini par $E_{<+\infty} = \{e \in E | \theta(e) < +\infty\}$ est une configuration.*

Démonstration. $E_{<+\infty}$ est causalement clos : soit $e \in E_{<+\infty}$ et soit $e' < e$. Par le lemme 5, on a $\theta(e') \leq \theta(e)$. Or $\theta(e) < +\infty$, donc $\theta(e') < +\infty$ et finalement $e' \in E_{<+\infty}$.

$E_{<+\infty}$ est sans conflit : procédons par l'absurde. Si $\#E_{<+\infty}$, alors, d'après le lemme 3 :

- soit $\mathsf{RCycles}(E_{<+\infty}) \neq \emptyset$, mais c'est interdit par l'équation 4.3 qui impose qu'au moins un des éléments de chaque cycle ne soit pas dans $E_{<+\infty}$;
- soit $\exists x,y \in E_{<+\infty}$ t.q. $x \,\mathsf{conf}\, y$. Par définition de $E_{<+\infty}$, on a $\theta(x) \neq +\infty$ et $\theta(y) \neq +\infty$, ce qui contredit l'équation 4.5.

□

Lemme 7. *Soit $(\mathcal{B},\theta)$ un TBP valide d'un TPN et E son ensemble d'événements. Soit $e \in E$ tel que $\theta(e) \neq +\infty$. Alors :*

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

Démonstration. Procédons par l'absurde. Si $\theta(e') < \theta(e)$ alors $e' \in \mathsf{Earlier}(e)$. Donc, par définition de Cut, chaque condition étant produite une fois et une seule, aucune condition de ${}^\bullet e'$ n'appartient à $\mathsf{Cut}(\mathsf{Earlier}(e))$, ce qui contredit ${}^\bullet e' \cap \mathsf{Cut}(\mathsf{Earlier}(e)) \neq \emptyset$.

□

Théorème 6. *Soit $\mathcal{N}$ un réseau de Petri temporel. Soit $(\mathcal{B},\theta)$ un processus de branchement temporel temporellement complet de $\mathcal{N}$ Soit $E_{<+\infty} = \{e \in E \text{ t.q. } \theta(e) < \infty\}$ et $\theta_{<+\infty}$ la restriction de θ à $E_{<+\infty}$.*

$(E_{<+\infty},\theta_{<+\infty})$ est un processus temporel valide de $\mathcal{N}$.

Démonstration. Par le lemme 6, on sait déjà que $E_{<+\infty}$ est une configuration et donc que $(E_{<+\infty},\theta)$ est un processus temporel. On montre donc que ce processus temporel est valide.

Soit $e \in E_{<+\infty}$. Par définition, on a $\theta(e) \neq +\infty$.

L'équation 4.4 donne de façon immédiate la contrainte de l'équation 4.1. Il nous reste à montrer que $\forall t \in \mathsf{Enabled}(l(\mathsf{Cut}(\mathsf{Earlier}(e))), \theta(e) - \mathsf{TOE}(\mathsf{Cut}(\mathsf{Earlier}(e)),t) \in I_s(t)^\downarrow$.

Soit $t \in \mathsf{Enabled}(l(\mathsf{Cut}(\mathsf{Earlier}(e))))$ et soit $B' = \{b \in \mathsf{Cut}(\mathsf{Earlier}(e)) | l(b) \in {}^\bullet t \cup {}^\circ t\}$ le sous-ensemble de $\mathsf{Cut}(\mathsf{Earlier}(e))$ qui sensibilise t.

Si $t = l(e)$ alors, comme le réseau est sauf, $B' = {}^\bullet e \cup {}^\circ e$. Par ailleurs, d'après l'équation 4.4, $\theta(e) - \mathsf{TOE}({}^\bullet e \cup {}^\circ e, l(e)) \in I_s(l(e))^\downarrow$, ce que nous pouvons donc réécrire en $\theta(e) - \mathsf{TOE}(B',t) \in I_s(t)^\downarrow$ et l'on trouve bien la contrainte attendue.

Si $t \neq l(e)$ mais qu'il existe $e' \in E$ tel que $l(e') = t$ et ${}^\bullet e' \cup {}^\circ e' = B'$ alors :

- Si $\theta(e') \neq +\infty$ alors, d'après l'équation 4.4, $\theta(e') - \mathsf{TOE}(B',t) \in I_s(t)^\downarrow$. Et avec le lemme 7, puisque ${}^\bullet e' \cap \mathsf{Cut}(\mathsf{Earlier}(e)) \neq \emptyset$, on a $\theta(e) \leq \theta(e')$, ce qui nous donne le résultat attendu.
- Si $\theta(e') = +\infty$ alors l'une des deux équations 4.7 ou 4.8 est satisfaite. Mais puisque ${}^\bullet e' \cup {}^\circ e' = B' \subseteq \mathsf{Cut}(\mathsf{Earlier}(e))$, chacune des conditions de B' a une date de production finie, ce qui exclut l'équation 4.7. Donc il existe $e'' \in E$ t.q. soit $e' \,\mathsf{conf}\, e''$, soit $e' \nearrow e''$, et $\theta(e'') \neq +\infty$ et $\theta(e'') - \mathsf{TOE}(B',t) \in I_s(t)^\downarrow$. On ne peut pas avoir $e' < e''$ car $\theta(e') = +\infty$ et cela contredirait le lemme 5. Donc soit ${}^\bullet e' \cap {}^\bullet e'' \neq \emptyset$, soit ${}^\circ e' \cap {}^\bullet e'' \neq \emptyset$. Donc ${}^\bullet e'' \cap \mathsf{Cut}(\mathsf{Earlier}(e)) \neq \emptyset$ et avec le lemme 7, $\theta(e) \leq \theta(e'')$, ce qui nous donne le résultat attendu.

Si aucun événement de E ne correspond au tir de t, c'est donc qu'il existe une extension possible (t,M,M') de $\mathcal{B}$ telle que $M \cup M' \subseteq \mathsf{Cut}(\mathsf{Earlier}(e))$. Dans ce cas, puisque $\mathcal{B}$ est temporellement complet, alors par la definition 38, soit directement $\theta(e) - \mathsf{TOE}(M \cup M',t) \in I_s(t)^\downarrow$, soit $\exists e'$ t.q. $\theta(e') - \mathsf{TOE}(M \cup M',t) \in I_s(t)^\downarrow$ et ${}^\bullet e' \cap (M \cup M') \neq \emptyset$. Alors, avec le lemme 7, on a $\theta(e) \leq \theta(e')$ et le résultat attendu.

□

Théorème 7. *Soit $\mathcal{N}$ un réseau de Petri temporel. Soit (C,θ) un processus temporel valide de $\mathcal{N}$. C est une configuration du processus de branchement $\mathcal{B} = (C^\bullet,C,F,F_r,m_0)$ de $\mathsf{Untimed}(\mathcal{N})$.*

$(\mathcal{B},\theta)$ est un processus de branchement temporel temporellement complet.

Démonstration. Puisque C est une configuration, on a bien $\mathsf{RCycles}(C) = \emptyset$. Donc l'équation 4.3 est vérifiée.

Soit maintenant e un événement de C.

(C,θ) étant un processus temporel, on a forcément $\theta(e) \neq \infty$ (ce qui satisait donc les contraintes des équations 4.7 et 4.8 par défaut).

L'équation 4.4 est trivialement vérifiée grâce aux contraintes des équations 4.1 et 4.2.

Puisque C est une configuration, il n'y a pas d'événements en conflit direct donc l'équation 4.5 est vérifiée.

Finalement, soit $e' \in C$ t.q. $e \nearrow e'$ et $e' \in C$. Si $e < e'$ alors, puisque (C,θ) est valide, le lemme 4 nous donne $\theta(e) \leq \theta(e')$. Sinon, ${}^\circ e \cap {}^\bullet e' \neq \emptyset$ et si $\theta(e') < \theta(e)$ alors des conditions ${}^\bullet e$ sont consommées avant que e se produise, ce qui est impossible car le réseau est sauf. Donc $\theta(e) \leq \theta(e')$.

On prouve maintenant que $(\mathcal{B},\theta)$ est temporellement complet. Soit (t,M,M') une extension possible de $\mathcal{B}$. On a forcément $M \cup M' \subseteq \mathsf{Cut}(C)$ et donc $t \in \mathsf{Enabled}(l(\mathsf{Cut}(C)))$. Donc il n'existe aucun événement $e' \in C$ tel que ${}^\bullet e' \cap (M \cup M') \neq \emptyset$ et il faut donc prouver que $\max\{\theta(e'')|e'' \in E, \theta(e'') \neq +\infty\} - \mathsf{TOE}(M \cup M',t) \in I_s(t)^\downarrow$. Soit e l'événement qui réalise ce maximum et soit e' l'événement ayant produit la condition la plus récente dans $M \cup M'$. Si $\theta(e') < \theta(e)$ alors $t \in \mathsf{Enabled}(l(\mathsf{Cut}(\mathsf{Earlier}(e))))$ et l'équation 4.2 permet de conclure. Sinon, si $\theta(e) < \theta(e')$ alors e ne réalise pas le maximum des éléments de C pour θ. Et finalement, si $\theta(e) = \theta(e')$ alors $\theta(e) = \mathsf{TOE}(M \cup M',t)$ et donc $\theta(e) - \mathsf{TOE}(M \cup M',t) \in I_s(t)^\downarrow$. □

Proposition 3. *L'ensemble D des fonctions de temporisation d'un processus de branchement temporel symbolique fini (dont le processus de branchement sous-jacent est fini) $(\mathcal{B},D)$ peut être représenté par une union finie de zones à coefficients rationnels.*

Démonstration. Soit $\mathcal{B}$ un processus de branchement fini. L'ensemble des contraintes sur les fonctions de temporisation du processus de branchement temporel symbolique correspondant est donné par la définition 37.

On peut représenter cet ensemble par une union finie (car le nombre d'événements est fini) d'ensembles de contraintes sur les dates d'occurrences finies des événements : les variables n'apparaissant pas dans l'un de ces ensembles ont implicitement une date d'occurrence infinie dans cet ensemble.

Puis chacun de ces ensembles peut être représenté par une union de zones à coefficients rationnels : il suffit pour cela de tranformer les contraintes du type $\theta(e) - \max\{\theta(e_1),\theta(e_2)\} \leq k$ en la disjonction de conjonctions $(\theta(e_1) \leq \theta(e_2)$ et $\theta(e) - \theta(e_2) \leq k)$ ou $(\theta(e_1) > \theta(e_2)$ et $\theta(e) - \theta(e_1) \leq k)$. De même, les contraintes du type $\theta(e) - \max\{\theta(e_1),\theta(e_2)\} \geq k$ donne la conjonction $(\theta(e) - \theta(e_1) \geq k$ et $\theta(e) - \theta(e_1) \geq k)$. Ces deux transformations se généralisent évidemment à un nombre fini quelconque d'éléments dans le max. □

Lemme 8. *Soient (C,θ) et (C',θ') deux processus temporels valides et futurs-équivalents d'un TPN $\mathcal{N}$. Soient $t \in \mathsf{Enabled}(l(\mathsf{Cut}(C))) = \mathsf{Enabled}(l(\mathsf{Cut}(C')))$ et b'^* la condition telle que $\mathsf{TOE}(\mathsf{Cut}(C'),t) = \theta'(b'^*)$. Soit $\delta' = \delta - \max_{b \in \mathsf{Cut}(C)} \theta({}^\bullet b) + \max_{b \in \mathsf{Cut}(C')} \theta'({}^\bullet b)$.*

Si $\mathsf{age}(b'^*,\theta',\mathsf{Cut}(C')) \neq K^*(b'^*)$ alors $\delta - \mathsf{TOE}(\mathsf{Cut}(C),t) = \delta' - \mathsf{TOE}(\mathsf{Cut}(C'),t)$.

Démonstration. Soit b^* la condition de $\mathsf{Cut}(C)$ telle que $l(b^*) = l(b'^*)$. Comme (C,θ) et (C',θ') sont futurs-équivalents, on a $\forall b \in \mathsf{Cut}(C), \forall b' \in \mathsf{Cut}(C')$, t.q. $l(b) = l(b'), \mathsf{age}(b,\theta,\mathsf{Cut}(C)) = \mathsf{age}(b',\theta',\mathsf{Cut}(C'))$. Donc, l'âge de b'^* n'étant pas « réduit », $\max_{b'' \in \mathsf{Cut}(C)} \theta(\bullet b'') - \theta(\bullet b^*) = \max_{b'' \in \mathsf{Cut}(C')} \theta(\bullet b'') - \theta(\bullet b'^*)$. Donc, par définition de δ', on a $\delta - \mathsf{TOE}(\mathsf{Cut}(C),t) = \delta' - \mathsf{TOE}(\mathsf{Cut}(C'),t)$. □

Théorème 8. *Soient (C,θ) et (C',θ') deux processus temporels valides et futurs-équivalents d'un TPN $\mathcal{N}$. Soit (t,M,M_r) une extension possible de C telle qu'il existe une extension possible (t,M',M'_r) de C'.*

Alors pour tout $\delta \geq \max_{b \in \mathsf{Cut}(C)} \theta(\bullet b)$ tel que (C_t,θ_δ) est valide, $(C'_t,\theta'_{\delta'})$ est valide pour $\delta' = \delta - \max_{b \in \mathsf{Cut}(C)} \theta(\bullet b) + \max_{b \in \mathsf{Cut}(C')} \theta'(\bullet b)$.

De plus, (C_t,θ_δ) et $(C'_t,\theta'_{\delta'})$ sont futur-équivalents.

Démonstration. On s'intéresse d'abord à la validité de $(C'_t,\theta'_{\delta'})$:

1. Commençons par montrer que $(C'_t,\theta'_{\delta'})$ satisfait la contrainte de l'équation 4.1 : $\delta' - \mathsf{TOE}(\mathsf{Cut}(C'),t) \in I_s(t)^\uparrow$. Soit b'^* la condition de $\mathsf{Cut}(C')$ telle que $\mathsf{TOE}(\mathsf{Cut}(C'),t) = \theta'(b'^*)$. Si $\mathsf{age}(b'^*,\theta',\mathsf{Cut}(C')) \neq K^*(b'^*)$, alors d'après le lemme 8, c'est équivalent à $\delta - \mathsf{TOE}(\mathsf{Cut}(C),t) \in I_s(t)^\uparrow$, ce qui est vrai car (C_t,θ_δ) est valide.
   Si $\mathsf{age}(b'^*,\theta',\mathsf{Cut}(C')) = K^*(b'^*)$, alors, par définition de l'âge de b'^*, $\mathsf{TOE}(\mathsf{Cut}(C'),t) \leq \max_{b \in \mathsf{Cut}(C')} \theta'(\bullet b) - K^*(b'^*)$. Donc $\delta' - \mathsf{TOE}(\mathsf{Cut}(C'),t) \leq \delta' - \max_{b \in \mathsf{Cut}(C')} \theta(\bullet b) + K^*(b'^*)$ et, par définition de δ', $\delta' - \mathsf{TOE}(\mathsf{Cut}(C'),t) \leq \delta - \max_{b \in \mathsf{Cut}(C)} \theta(\bullet b) + K^*(b'^*)$. Or, par définition, $K^*(b'^*) \geq \min I_s(t)$ et, par hypothèse, $\delta - \max_{b \in \mathsf{Cut}(C)} \theta(\bullet b) \geq 0$ donc $\delta' - \mathsf{TOE}(\mathsf{Cut}(C'),t) \in I_s(t)^\downarrow$.
2. Prouvons maintenant que $(C'_t,\theta'_{\delta'})$ satisfait la contrainte de l'équation 4.2 est : $\forall t' \in \mathsf{Cut}(\mathsf{Earlier}((t,M',M'_r))), \delta' - \mathsf{TOE}(\mathsf{Cut}(C'),t') \in I_s(t')^\downarrow$.
   - si $\delta' = \mathsf{TOE}(\mathsf{Cut}(C'),t)$ alors $\mathsf{Cut}(\mathsf{Earlier}((t,M',M'_r))) = \mathsf{Cut}(\mathsf{Earlier}(\bullet b))$ pour b étant la plus récente condition de $M' \cup M'_r$. Et comme (C',θ') est valide, on a bien la validité de $(C'_t,\theta_{\delta'})$.
   - si $\delta' > \mathsf{TOE}(\mathsf{Cut}(C'),t)$ alors $\mathsf{Cut}(\mathsf{Earlier}((t,M',M'_r))) = \mathsf{Cut}(C')$ et l'on a de même $\mathsf{Cut}(\mathsf{Earlier}((t,M,M_r))) = \mathsf{Cut}(C)$. Considérons la condition b'^* de $\mathsf{Cut}(C')$ telle que $\mathsf{TOE}(\mathsf{Cut}(C'),t) = \theta'(b'^*)$. Si $\mathsf{age}(b'^*,\theta',\mathsf{Cut}(C')) \neq K^*(b'^*)$, par le lemme 8, la condition à prouver est équivalente à $\delta - \mathsf{TOE}(\mathsf{Cut}(C),t') \in I_s(t)^\downarrow$ ce qui est vrai car (C,θ) est valide.
     Si $\mathsf{age}(b'^*,\theta',\mathsf{Cut}(C')) = K^*(b'^*)$ alors notons b^* la condition de $\mathsf{Cut}(C)$ telle que $l(b^*) = l(b'^*)$. Comme les deux processus sont futur-équivalents, $\mathsf{age}(b^*,\theta,\mathsf{Cut}(C)) = K^*(b^*)$ et donc, par définition de l'âge de b^*, $\mathsf{TOE}(\mathsf{Cut}(C),t') \leq \max_{b \in \mathsf{Cut}(C)} \theta(\bullet b) - K^*(b^*)$. Donc $\delta - \mathsf{TOE}(\mathsf{Cut}(C),t') \geq \delta - \max_{b \in \mathsf{Cut}(C)} \theta(\bullet b) + K^*(b^*)$. Et donc, comme par hypothèse $\delta \geq \max_{b \in \mathsf{Cut}(C)} \theta(\bullet b)$, $\delta - \mathsf{TOE}(\mathsf{Cut}(C),t') \geq K^*(b^*)$. Comme (C,θ) est valide, cela implique que $I_s(t')$ est ouvert à droite en l'infini sinon la borne maximale de l'intervalle est dépassée. Ce qui nous permet de conclure que $\delta' - \mathsf{TOE}(\mathsf{Cut}(C'),t') \in I_s(t')^\downarrow$.

Montrons enfin que (C_t,θ_δ) et $(C'_t,\theta'_{\delta'})$ sont futur-équivalents. On a vu qu'on avait bien $l(\mathsf{Cut}(C_t)) = l(\mathsf{Cut}(C'_t))$. Soient $b \in \mathsf{Cut}(C_t)$ et $b' \in \mathsf{Cut}(C'_t)$ telle que $l(b) = l(b')$. Si $l(b) \in t^\bullet$ alors, comme $\delta \geq \max_{b'' \in \mathsf{Cut}(C)} \theta(\bullet b'')$, on a $\max_{b'' \in \mathsf{Cut}(C_t)} \theta_\delta(\bullet b'') = \theta(b)$ et donc $\mathsf{age}(b,\theta_\delta,\mathsf{Cut}(C_t)) = 0$. Et, par définiton de δ', on a de la même façon $\max_{b'' \in \mathsf{Cut}(C'_t)} \theta'_{\delta'}(\bullet b'') = \theta(b')$ et donc $\mathsf{age}(b',\theta'_{\delta'},\mathsf{Cut}(C'_t)) = 0$.

Si $l(b) \notin t^\bullet$ alors $l(b) \notin {}^\bullet t$ sinon $b \notin \mathsf{Cut}(C_t)$. Donc, $\mathsf{age}(b,\theta_\delta,\mathsf{Cut}(C_t)) = \mathsf{age}(b,\theta,\mathsf{Cut}(C)) + \delta - \max_{b'' \in \mathsf{Cut}(C)} \theta(\bullet b'')$. Et, de façon similaire, $\mathsf{age}(b',\theta'_{\delta'},\mathsf{Cut}(C'_t)) = \mathsf{age}(b',\theta',\mathsf{Cut}(C')) + \delta' -$

$\max_{b'' \in \mathsf{Cut}(C')} \theta'(^\bullet b'')$. D'où, par définition de δ', le résultat attendu : $\mathsf{age}(b,\theta_\delta,\mathsf{Cut}(C_t)) = \mathsf{age}(b',\theta'_{\delta'},\mathsf{Cut}(C'_t))$. □

Proposition 4. *Soit $\mathcal{N}$ un TPN. Pour tout processus temporel valide (C,θ) de $\mathcal{N}$ et tout événement $e \in C$, $(\mathsf{Past}((C,\theta),e),\theta)$ est un processus temporel valide de $\mathcal{N}$.*

Démonstration. Soit $e \in C$. Afin de simplifier l'écriture, on note $\mathsf{Past}(e)$ au lieu de $\mathsf{Past}((C,\theta),e)$.

1. Prouvons d'abord que $\mathsf{Past}(e)$ est une configuration. C est sans conflit donc $\mathsf{Past}(e) \subseteq C$ l'est également. Soit $e' \in \mathsf{Past}(e)$. Supposons d'abord $e \in \lceil e \rceil$. Alors comme $\lceil e \rceil$ est fermé par causalité, on a bien $\forall e'' < e', e'' \in \mathsf{Past}(e)$. Supposons maintenant $e' \in \mathsf{Earlier}(e)$. Soit $e'' < e'$. On a alors $\theta(e'') \leq \theta(e')$ et $\theta(e') < \theta(e)$. Donc $\theta(e'') < \theta(e)$. Donc $e'' \in \mathsf{Earlier}(e) \subseteq \mathsf{Past}(e)$. Donc $\mathsf{Past}(e)$ est bien fermé par causalité.
2. Prouvons maintenant que $(\mathsf{Past}(e),\theta)$ est valide. Soit $e' \in \mathsf{Past}(e)$.
   - La contrainte de l'équation 4.1 est vérifiée pour e' dans (C,θ), elle l'est toujours pour e' dans $(\mathsf{Past}(e),\theta)$.
   - Soit $t \in \mathsf{Enabled}(l(\mathsf{Cut}(\mathsf{Earlier}((\mathsf{Past}(e),\theta),e'))))$. On prouve que $\mathsf{Earlier}((C,\theta),e') = \mathsf{Earlier}((\mathsf{Past}(e),\theta),e')$. $\mathsf{Past}(e) \subseteq C$ donc on a trivialement $\mathsf{Earlier}((\mathsf{Past}(e),\theta),e') \subseteq \mathsf{Earlier}((C,\theta),e')$. Prouvons alors l'inclusion inverse. Soit $e'' \in \mathsf{Earlier}((C,\theta),e')$. $\theta(e'') < \theta(e')$ et $\theta(e') \leq \theta(e)$ car, soit $e' \in \mathsf{Earlier}((C,\theta),e)$, soit $e' < e$. Donc $\theta(e'') < \theta(e)$. Par conséquent, $e'' \in \mathsf{Past}(e)$ et donc $e'' \in \mathsf{Earlier}((\mathsf{Past}(e),\theta),e')$. On en déduit que $t \in \mathsf{Enabled}(l(\mathsf{Cut}(\mathsf{Earlier}((C,\theta),e'))))$ et comme (C,θ) est valide, on a le résultat attendu.

□

Théorème 9. *Soit $\mathcal{N}$ un TPN. Pour tout processus temporel fini et valide (C,θ) de $\mathsf{Unfolding}(\mathcal{N})$, il existe un processus temporel fini et valide (C',θ') dans $CFP_\prec(\mathcal{N})$ tel que (C,θ) et (C',θ') sont futur-équivalents.*

Démonstration. On reprend le schéma de preuve de [43].

Soit (C,θ) un processus temporel fini et valide de $\mathsf{Unfolding}(\mathcal{N})$. Si (C,θ) est un processus de $CFP_\prec(\mathcal{N})$ alors on a bien le résultat attendu. Supposons donc que (C,θ) n'est pas dans $CFP_\prec(\mathcal{N})$. Alors il existe un événement d'arrêt $e \in C$ et donc un autre événement e' tels que $\lceil e' \rceil \prec \lceil e \rceil$ et $l(\mathsf{Cut}(\lceil e' \rceil)) = l(\mathsf{Cut}(\lceil e \rceil))$. Il existe de plus un processus temporel $(C',\theta') \in \mathcal{P}(e')$ tel que $(\mathsf{Past}((C,\theta),e),\theta)$ et $(\mathsf{Past}((C,\theta'),e'),\theta')$ sont futur-équivalents.

Par ailleurs, on sait grâce à la proposition 4 que ces deux processus sont valides et l'on peut donc leur appliquer le théorème 8 : puisque $(\mathsf{Past}(e),\theta)$ s'étend, en ajoutant les événements par ordre croissant de dates d'occurrences, en (C,θ) qui est valide, on peut étendre $(\mathsf{Past}(e'),\theta')$ en (C'',θ'') qui est donc également valide et futur-équivalent à (C,θ).

De plus, comme $l(\mathsf{Cut}(\lceil e' \rceil)) = l(\mathsf{Cut}(\lceil e \rceil))$, $\lceil e' \rceil$ et $\lceil e \rceil$ s'étendent respectivement en C'' et C par deux séquences d'extensions concernant les mêmes transitions. Comme $\prec$ est préservé par extension, on a alors $C'' \prec C$. Si (C'',θ'') n'est pas dans $CFP_\prec(\mathcal{N})$, on peut itérer ce raisonnement. On obtient ainsi une suite de processus temporels valides $(C_1,\theta_1),\ldots,(C_n,\theta_n)$, tous futur-équivalents, et tels que $(C_1,\theta_1) = (C,\theta)$ et $C_n \prec \cdots \prec C_1$.

Comme $\prec$ est bien fondé on sait que cette suite est nécessairement finie et que donc (C_n,θ_n) est dans $CFP_\prec(\mathcal{N})$. □

Théorème 10. *Soit $\mathcal{N}$ un TPN. $CFP_\prec(\mathcal{N})$ est fini.*

Démonstration. Soient e un événement de $\mathsf{Unfolding}(\mathcal{N})$ et (C,θ) un processus temporel fini et valide tel que $e \in C$. Le nombre de valeurs possibles pour $l(\mathsf{Cut}(C))$ est fini car $\mathcal{N}$ est sauf. Pour chaque place p de $\mathcal{N}$, on se donne une variable correspondante x_p. Il y en a donc un nombre fini.

D'après la proposition 3, pour le processus de branchement temporel symbolique fini (C,D), D peut être représenté par une union finie de zones à coefficients rationnels. À partir de D, on peut obtenir les âges possibles des jetons de $l(\mathsf{Cut}(\mathsf{Past}((C,\theta),e))$, pour tous les $\theta \in D$, par les changements de variables $\forall b \in \mathsf{Cut}(C), \theta(^\bullet b)$ devient $x_{l(b)}$ et l'élimination des autres variables. On obtient alors l'âge réduit des jetons en tronquant les bornes supérieures. Ces transformations classiques donnent à nouveau une union finie de zones [27]. Il est par ailleurs facile de voir que les dénominateurs des coefficients des zones sont bornés (par le PPCM des bornes finies des intervalles statiques).

On en déduit que l'union $A(e)$ pour tous les processus temporels finis (C,θ) contenant e des unions finies de zones représentant les âges possibles des jetons de $l(\mathsf{Cut}(\mathsf{Past}((C,\theta),e)))$ peut encore être représentée par une union *finie* de zones à coefficients rationnels bornés (et de dénominateurs bornés) car il n'y a qu'un nombre fini de zones répondant à ces critères. Et donc $A(e)$ ne peut prendre qu'un nombre fini de valeurs sur l'ensemble des événements.

Si $CFP_\prec(\mathcal{N})$ est infini, alors il contient nécessairement une séquence infinie d'événements $e_1 < e_2 < \cdots < e_n < \cdots$. Et parmi ces événements il existe forcément deux événements e et e' tels que $e' < e$, $l(\mathsf{Cut}(\lceil e \rceil)) = l(\mathsf{Cut}(\lceil e' \rceil))$ et $A(e) = A(e')$. Alors, pour tout processus temporel fini et valide (C,θ) tel que $e \in C$, l'âge des jetons de $l(\mathsf{Cut}(\mathsf{Past}(C,\theta),e))$ est une solution de $A(e)$ et donc de $A(e')$. Donc il existe un processus temporel fini et valide (C',θ') tel que $e' \in C'$ et $(\mathsf{Past}((C,\theta),e),\theta)$ et $(\mathsf{Past}((C',\theta'),e'),\theta')$ sont futur-équivalents (la solution commune implique que les variables sont les mêmes et donc $l(\mathsf{Cut}(\mathsf{Past}((C,\theta),e))) = l(\mathsf{Cut}(\mathsf{Past}((C,\theta),e))))$. Par ailleurs, $e' < e$ donc $\lceil e' \rceil \subseteq \lceil e \rceil$, et puisque $\prec$ raffine l'ordre préfixe, on a $\lceil e' \rceil \prec \lceil e \rceil$. Et donc e est un événement d'arrêt, ce qui contredit la définition de $CFP_\prec(\mathcal{N})$ qui est donc nécessairement fini. □