

# THÈSE

présentée à l'École Normale Supérieure de Cachan

pour obtenir le grade de

**Docteur de l'École Normale Supérieure de Cachan**

par : Pierre-Alain REYNIER

Spécialité : INFORMATIQUE

**Vérification de systèmes temporisés et distribués :  
modèles, algorithmes et implémentabilité.**

Soutenue le 21 juin 2007, devant un jury composé de :

- |                        |                        |
|------------------------|------------------------|
| – Parosh Aziz ABDULLA  | examineur              |
| – Ahmed BOUAIJANI      | rapporteur             |
| – Patricia BOUYER      | co-directrice de thèse |
| – Serge HADDAD         | examineur              |
| – Claude JARD          | examineur              |
| – François LAROUSSINIE | co-directeur de thèse  |
| – Jean-François RASKIN | rapporteur             |



## Résumé

Avec le développement de l'informatique, le nombre et le rôle des systèmes critiques augmentent constamment. Une proportion importante de ces systèmes présente des aspects temporisés exprimant des contraintes quantitatives sur l'écoulement du temps. Les travaux présentés dans cette thèse s'inscrivent dans le cadre de la vérification automatique de systèmes temporisés et distribués. Nous considérons différents modèles obtenus comme des extensions des automates temporisés et des réseaux de Petri. Dans une première partie, nous étudions ces modèles à la fois indépendamment, obtenant de nouveaux résultats d'indécidabilité pour les automates temporisés avec transitions silencieuses, et en s'intéressant aux liens existant entre eux. Nous proposons ainsi une traduction des automates temporisés étendus vers les réseaux de Petri temporels et comparons avec précision les pouvoirs expressifs des automates temporisés et des réseaux de Petri temporisés. La deuxième partie de nos travaux traite de problèmes d'algorithmique de la vérification. Nous proposons un algorithme pour l'analyse en avant des automates temporisés avec gardes diagonales ainsi qu'une notion de dépliage pour les réseaux d'automates temporisés et différentes méthodes pour en calculer un préfixe fini et complet. Enfin, dans une dernière partie, nous nous intéressons à la notion d'implémentabilité pour les automates temporisés et proposons un algorithme pour la vérification robuste de la logique temporelle linéaire.

**Mots-Clefs :** Vérification formelle, Automates temporisés, Réseaux de Petri, Systèmes distribués, Expressivité, Algorithmique, Implémentabilité.

## Abstract

With the development of computer science, the number and the role of critical systems always grow. A large part of these systems presents timing aspects expressing some quantitative constraints on time elapsing. The results presented in this thesis fall into the framework of automated verification of distributed and timed systems. We consider different models obtained as extensions of timed automata and Petri nets. In a first part, we study these models both independently, proving new undecidability results for timed automata with silent transitions, and by looking at links between them. We propose a translation from extended timed automata to time Petri nets, and compare relative expressive power of timed automata and timed Petri nets. The second part of our work deals with problems in algorithmic of verification. We propose an algorithm for forward analysis of timed automata with diagonal constraints, and a notion of unfolding for networks of timed automata together with different methods for building a finite and complete prefix of it. Finally, we are interested in a notion of implementability for timed automata and propose an algorithm for robust verification of linear temporal logic.

**Keywords :** Formal verification, Timed automata, Petri nets, Distributed systems, Expressivity, Algorithmic, Implementability.



# Remerciements

Je tiens à remercier Parosh Aziz Abdulla, Ahmed Bouajjani, Serge Haddad, Claude Jard et Jean-François Raskin d'avoir accepté de faire partie de mon jury et d'avoir été présent lors de la soutenance. Special thanks go to Parosh Aziz Abdulla who came from Sweden especially for my PhD defense. Je remercie aussi plus particulièrement Ahmed Bouajjani et Jean-François Raskin, les rapporteurs de ma thèse, qui ont relu mon manuscrit malgré sa longueur, et Claude Jard, qui a présidé le jury.

Mes remerciements les plus vifs vont ensuite à mes directeurs de thèse, Patricia Bouyer et François Laroussinie, qui tous deux ont énormément compté dans la réussite de cette thèse. Dès mon arrivée à l'ENS, lorsqu'ils étaient mes enseignants, ils ont su me donner le goût de l'informatique théorique. Puis, pendant mes trois années de thèse, avec une parfaite complémentarité, leurs nombreuses qualités (en plus des évidentes qualités scientifiques), parmi lesquelles je mentionne (sans espérer être exhaustif !) la simplicité, la bonne humeur, la disponibilité, la curiosité et l'écoute, m'ont permis de réaliser ma thèse dans d'extraordinaires conditions de travail.

Je souhaite également remercier mes autres co-auteurs, Serge Haddad et Nicolas Markey, avec qui j'ai eu la chance et le plaisir de travailler. Ils ont su par leur constante ingéniosité m'aider à avancer et me donner confiance en moi, bref me former à la recherche !

Cette thèse a été réalisée au Laboratoire Spécification et Vérification, et cela constitue sûrement un des facteurs de réussite. Ce laboratoire constitue un cadre favorable pour faire de la recherche, alliant des qualités scientifiques exceptionnelles et une grande convivialité. Je remercie donc l'ensemble des membres du laboratoire pour leurs conseils avisés au cours de ces trois années et pour les moments de détente autour des goûters et des pots.

Ces trois années, je les ai passées dans la salle Renaudeau, sorte d'ilôt chaleureux de thésards au sein du labo. Alternier les xblast avec les préparations de TPs, les débats politiques passionnés avec les discussions (tout autant passionnées ?) sur les automates temporisés, une sortie piscine ou un foot avec un rush de deadline, tout cela a permis de rendre le travail nettement plus agréable ! Alors pour tout ça, je remercie les membres de Renaudeau, les anciens comme les nouveaux : par ordre alphabétique j'ai pu y croiser Arnaud, Christophe, Fabrice, Florent, Jean-Loup, Jules, Julien, Najla, Reggi, Rémi et Seb. C'est là que je me rends compte que ça fait beaucoup de monde !

Mais il n'y a pas que le labo, bien sûr, et les amis ont beaucoup compté pour tenir le coup. Pendant ces trois années, Ronan, TT, Adrien et Virginie ont été des partenaires infailibles des soirées coinche, quasi hebdomadaires. Merci beaucoup à vous tous pour ces bons moments. Bon, notez que vous auriez pu me laisser gagner plus souvent ! Merci aussi à Thomas qui n'a jamais rechigné à venir faire un tour dans le Champsaur quand on était à Pisançon, et à Houssam qui lui aussi a pas mal voyagé pour qu'on puisse se voir, entre Grenoble, Strasbourg, Paris et plus au nord encore...

Enfin, j'aimerais remercier ma famille pour son soutien moral et son accompagnement depuis toujours, et en particulier mes parents qui m'ont donné un goût certain pour les études et m'ont

permis d'arriver jusque là. Merci aussi pour le sacré coup de main pour l'organisation du pot, des rillettes sarthoises aux fromages arrivés en livraison express du Champsaur !

Pour finir, je veux remercier celle qui entre-temps est devenue ma femme, et qui a accepté de faire certains sacrifices pour me permettre de faire ma thèse dans ces excellentes conditions, comme enseigner en banlieue parisienne dans des établissements "sensibles". Merci aussi pour les relectures du manuscrit, les conseils de rédaction pour l'intro, le soutien quotidien au long de ces années, mais encore et surtout merci pour tout ce que tu es chaque jour !

*À celle qui entretemps est devenue ma femme,  
Virginie.*





# Table des matières

<b>Table des matières</b>	<b>9</b>
<b>Introduction</b>	<b>13</b>
Enjeux de la vérification	13
Méthodes formelles	14
Vérification de modèles	15
Modélisation : Des formalismes de description	16
Vérification : Des problèmes d’algorithmique	17
Implémentation : Du modèle au système réel	18
Modèles étudiés dans la thèse	18
Des modèles temporisés ...	18
... et aussi distribués	19
Impact de la combinaison des deux aspects	19
Contributions de la thèse	20
Étude des formalismes de modélisation	20
Algorithmique de la Vérification	22
Implémentabilité	23
Plan de la thèse	23
<b>I Modèles Temporisés</b>	<b>25</b>
<b>1 Automates temporisés</b>	<b>27</b>
1.1 Préliminaires	27
1.1.1 Quelques éléments de mathématiques	27
1.1.2 Alphabet, mots temporisés et mots de délais	28
1.1.3 Horloges, valuations, remises à zéro et contraintes d’horloges	29
1.1.4 Intervalles de temps	31
1.2 Systèmes de transitions temporisés	31
1.2.1 Définitions	31
1.2.2 Composition	33
1.2.3 Comparaison	34
1.3 Automates temporisés d’Alur et Dill	35
1.4 Problème d’accessibilité, problème du vide	38
1.4.1 Pourquoi ces problèmes sont équivalents et fondamentaux	38
1.4.2 Décidabilité des automates temporisés	39

1.4.3	Automate des régions . . . . .	40
1.5	Extensions du modèle d'Alur et Dill . . . . .	41
1.5.1	Transitions silencieuses . . . . .	41
1.5.2	Mises à jour d'horloges . . . . .	42
1.5.3	Contraintes d'horloges diagonales . . . . .	44
1.6	Réseaux d'automates temporisés . . . . .	46
1.7	Outils existants . . . . .	51
<b>2</b>	<b>Extensions temporisées des réseaux de Petri</b>	<b>53</b>
2.1	Réseaux de Petri . . . . .	53
2.1.1	Préliminaires . . . . .	53
2.1.2	Définition . . . . .	54
2.1.3	Résultats classiques . . . . .	57
2.2	Réseaux de Petri temporels . . . . .	58
2.2.1	Définition . . . . .	58
2.2.2	Travaux existants . . . . .	60
2.3	Réseaux de Petri temporisés . . . . .	62
2.3.1	Définition . . . . .	62
2.3.2	Travaux existants . . . . .	64
2.4	Conclusion . . . . .	66
<b>II</b>	<b>Propriétés et Comparaison de Modèles Temporisés</b>	<b>67</b>
<b>3</b>	<b>Résultats d'indécidabilité pour les automates temporisés avec transitions silencieuses</b>	<b>69</b>
3.1	Introduction . . . . .	69
3.2	Préliminaires . . . . .	70
3.3	S'affranchir des transitions silencieuses . . . . .	72
3.4	Déterminisation, complémentation . . . . .	74
3.4.1	Cas d'un alphabet ayant trois lettres ou plus . . . . .	75
3.4.2	Cas d'un alphabet de deux lettres . . . . .	77
3.5	Minimisation du nombre d'horloges . . . . .	79
3.6	Opération de « mélange » . . . . .	82
3.7	Extension aux mots infinis . . . . .	85
3.8	Conclusion . . . . .	87
<b>4</b>	<b>Des automates temporisés étendus vers les réseaux de Petri temporels</b>	<b>89</b>
4.1	Introduction . . . . .	89
4.2	Cas d'un automate temporisé étendu . . . . .	91
4.2.1	Présentation de la construction . . . . .	91
4.2.2	Preuve de correction . . . . .	97
4.2.3	Résultats de complexité et de concision . . . . .	99
4.3	Cas d'un réseau d'automates temporisés avec invariants . . . . .	101
4.3.1	Prise en compte des invariants . . . . .	101
4.3.2	Synchronisation . . . . .	101
4.4	Conclusion . . . . .	102

<b>5 Réseaux de Petri temporisés et automates temporisés</b>	<b>105</b>
5.1 Introduction	105
5.2 Réseaux de Petri temporisés avec arcs de lecture	106
5.2.1 Définition du modèle	106
5.2.2 Équivalence avec les automates temporisés	109
5.3 Problème de couverture	113
5.4 Préliminaires à l'étude de l'expressivité	118
5.4.1 Deux langages temporisés discriminants	118
5.4.2 Forme normale pour les réseaux de Petri temporisés	121
5.5 Expressivité des arcs de lecture	127
5.5.1 Cas des mots finis	128
5.5.2 Cas des mots infinis non Zeno	134
5.6 Expressivité des mises à jour non déterministes	137
5.6.1 En absence d'arc de lecture	137
5.6.2 En présence d'arcs de lecture	138
5.7 Synthèse et applications	144
5.7.1 Synthèse des résultats d'expressivité	144
5.7.2 Applications aux automates temporisés	145
5.8 Conclusion	146
<b>III Algorithmes pour la Vérification</b>	<b>149</b>
<b>6 Analyse en avant des automates temporisés avec contraintes diagonales</b>	<b>151</b>
6.1 Introduction	151
6.2 Analyse en avant des automates temporisés	152
6.2.1 Les zones, une représentation symbolique de valuations	152
6.2.2 Matrices à différences bornées	156
6.2.3 Fondamentaux de la correction de l'algorithme	158
6.3 Analyse en avant et gardes diagonales	160
6.3.1 Illustration avec un contre-exemple	160
6.3.2 Zones, gardes diagonales et extrapolation	161
6.4 Méthodes pour corriger l'algorithme	163
6.4.1 S'affranchir des gardes diagonales	163
6.4.2 Détecter les faux-positifs	164
6.4.3 Raffinement «à la demande »	166
6.5 Analyse des contre-exemples	167
6.5.1 Définitions préliminaires	167
6.5.2 Quelques contre-exemples	168
6.5.3 Propagation des contraintes diagonales	169
6.5.4 Présentation et correction de la méthode	171
6.6 Implémentation dans UppAal	176
6.6.1 Description de la structure du programme	176
6.6.2 Bancs de tests	177
6.7 Conclusion	180

<b>7</b>	<b>Dépliage de réseaux d'automates temporisés</b>	<b>183</b>
7.1	Introduction . . . . .	183
7.2	Préliminaires . . . . .	187
7.2.1	Dépliage de réseaux d'automates finis . . . . .	187
7.2.2	Exemples explicatifs . . . . .	189
7.3	Dépliage non temporisé . . . . .	190
7.3.1	Réseau d'occurrences contextuel . . . . .	191
7.3.2	Processus branchant d'un automate temporisé . . . . .	193
7.4	Dépliage temporisé . . . . .	199
7.4.1	Processus non branchant temporisé . . . . .	199
7.4.2	Cohérence temporelle d'un processus non branchant temporisé . . . . .	202
7.4.3	Définition et construction du dépliage temporisé . . . . .	204
7.4.4	Construction du dépliage fondée sur des zones locales . . . . .	207
7.5	Préfixe fini et complet . . . . .	212
7.5.1	Introduction . . . . .	212
7.5.2	Automates bornés . . . . .	213
7.5.3	Événements synchronisés . . . . .	217
7.6	Discussion . . . . .	225
7.7	Conclusion . . . . .	228
<b>IV</b>	<b>De la vérification du modèle à l'implémentation du système</b>	<b>231</b>
<b>8</b>	<b>Implémentabilité et robustesse</b>	<b>233</b>
8.1	Introduction . . . . .	233
8.2	Vérification et implémentabilité . . . . .	235
8.2.1	Cadre de la sémantique « AASAP » . . . . .	236
8.2.2	Vérification « robuste » . . . . .	238
8.3	Vérification robuste de conditions co-Büchi . . . . .	241
8.4	Vérification robuste de LTL . . . . .	245
8.5	Vers la vérification robuste de MTL . . . . .	247
8.6	Conclusion . . . . .	250
	<b>Conclusion Générale</b>	<b>253</b>
	<b>Bibliographie</b>	<b>257</b>

# Introduction

Paris, bar des trois Moulins, 7h50.  
Mai 2007.

- Alors ?
- Hmm...
- Comment ça s'est fini ? Tu t'en es sorti ?
- Formaté, une fois de plus !
- Allez ! Fais pas cette tête, c'est pas si grave. Tu prends un café ?
- C'qui me dépasse c'est que l'informatique est partout aujourd'hui ! Tu prends ta voiture, le GPS te guide, tu vas à l'hôpital, tu passes un scanner, tu prends le Meteor, y'a plus de conducteur. Et attends, là j'ai passé deux heures à attendre pour pouvoir voter, et je suis même pas sûr que ma voix va compter...

## Enjeux de la vérification

Voilà la genre de conversations que l'on peut entendre partout de nos jours. L'informatique occupe une place prépondérante dans de nombreux domaines, tels la santé, les transports, l'économie, les communications et plus récemment le vote électronique. Et quand ces applications ont des enjeux importants comme des vies humaines, des moyens financiers ou encore la garantie de la démocratie, les systèmes informatiques doivent être irréprochables. Cependant, ces « systèmes critiques » présentent souvent des imperfections et l'Histoire a connu de nombreux faits divers qui en témoignent.

**Thérac-25.** Le Thérac-25 est un appareil médical utilisé aux États-Unis et au Canada à partir de 1976 pour traiter les tumeurs cancéreuses en exposant les patients à des radiations. Entre 1985 et 1987, date à laquelle tous les appareils ont été retirés, une série d'incidents ont eu lieu, provoquant la mort de plusieurs personnes. Une erreur logicielle a provoqué des surexpositions importantes, provoquant ainsi la mort de ces patients. Davantage de détails peuvent être trouvés dans [LT93].

**Ariane 5.** « Le vol inaugural d'Ariane 5 qui eut lieu le 4 juin 1996 s'est soldé par un échec. Environ 40 secondes seulement après le démarrage de la séquence de vol, le lanceur, qui se trouvait alors à une altitude de quelques 3700 mètres, a dévié de sa trajectoire, s'est brisé et a explosé. » Telles sont les deux premières phrases du rapport [Ari96] de la commission d'enquête mise en place suite à l'explosion d'Ariane 5. Celui-ci conclut à une erreur logicielle dans la spécification d'une fonction

(une valeur est codée sur 16 bits au lieu de 64 bits) et recommande le développement des méthodes de vérification.

**France Telecom.** Des milliers d'appels téléphoniques passés depuis la France sont restés sans réponse au cours des samedi 30 et dimanche 31 octobre 2004. Et pour cause, ces appels n'étaient pas traités par les commutateurs ! C'est l'analyse menée par les experts de France Telecom qui a permis de mettre en évidence une erreur logicielle sur un équipement de traitement de la voix sur IP situé à Reims. Son dysfonctionnement a ensuite provoqué un mécanisme d'autoprotection de vingt-six commutateurs, ralentissant ainsi une grande partie du réseau français. Les ingénieurs ont dû procéder au développement d'un logiciel de remplacement et l'ont implanté sur chacun des commutateurs, ce qui a nécessité l'ensemble du week-end.

**Vote électronique aux États-Unis.** Tandis que la France commence à utiliser les machines de vote électroniques, elles sont fortement remises en cause aux États-Unis. Un rapport complet [Bre06] sur les dangers du vote électronique a été publié par une commission d'experts du « Brennan Center », de l'Université de New York de Justice. Cette commission, composée de membres importants de la communauté scientifique comme David Dill et Ronald Rivest, émet des conclusions critiques concernant l'utilisation actuelle de ces machines, et propose un ensemble de mesures permettant d'augmenter la confiance dans ce système de vote. Plus récemment encore, en janvier 2007, une étude [Vot07] a recensé plus de 1000 dysfonctionnements au cours des élections de mi-mandat de 2006. Parmi ces dysfonctionnements, les logiciels sont plusieurs fois remis en cause, et le gouvernement américain a récemment décidé d'augmenter les tests de ces machines. Ainsi, un organisme de certification a perdu son accréditation et le code source pourrait être exigé auprès des fabricants.

## Méthodes formelles

Il apparaît primordial de pouvoir garantir une *sûreté de fonctionnement* de ces systèmes critiques. Les méthodes habituellement utilisées lors du développement de systèmes informatiques (comme des logiciels) consistent essentiellement à « tester » le fonctionnement du système dans un ensemble de situations correspondant au cadre d'utilisation « attendu », *i.e.* pour lequel le système a été conçu. Cependant, de telles méthodes ne permettent pas de tester le comportement du système dans *toutes* les situations, l'ensemble des scénarios étant le plus souvent infini. Une autre faiblesse de cette approche provient du manque de précision de ces tests. Les descriptions souvent informelles utilisées dans les cahiers des charges ne décrivent pas de façon satisfaisante les comportements admissibles et non-admissibles.

Bien que le test du système permette de détecter des erreurs, il est nécessaire de compléter les résultats obtenus par une autre approche. La solution que nous étudions dans ces travaux s'inscrit dans le cadre des *méthodes formelles*. L'idée principale de cette approche est de plonger la démarche de vérification du système dans un cadre formel afin de permettre l'utilisation de raisonnements mathématiques lors de l'analyse du système. De plus, le cadre mathématique permet une description du système et de ses comportements plus rigoureuse. Nous présentons quatre approches qui appartiennent à la classe des méthodes formelles, ce ne sont pas les seules, mais elles comptent parmi les plus importantes.

**Tests basés sur le modèle.** Dans cette approche, la notion de test est encore présente. Cependant, elle diffère du test « classique » par plusieurs aspects. D'abord, la propriété à tester ainsi que la partie fonctionnelle du système qui doit être analysée sont modélisées et décrites dans des

langages mathématiques. L'objectif consiste alors à générer de façon automatique un ensemble fini de scénarios de tests qui soit couvrant, *i.e.* qui assure que si le système « passe » ces tests, alors il vérifie la propriété. Malheureusement, dans de nombreux cas, un tel ensemble couvrant n'existe pas.

**Analyse statique.** Largement répandue dans les compilateurs, cette approche consiste à analyser certaines propriétés d'un code de façon statique, *i.e.* sans chercher à calculer l'ensemble des comportements possibles du système. Ceci permet par exemple de s'assurer qu'une variable est bien définie lors de son utilisation, ou encore qu'il n'existe pas d'accès à un tableau en dehors de son ensemble de définition si celui-ci est de taille constante. Cependant, les propriétés dynamiques ne peuvent pas être testées par cette approche.

**Démonstration automatique.** L'objectif est cette fois de bâtir un raisonnement logique démontrant la correction du système vis-à-vis d'une propriété. Cette correction est donc exprimée comme un théorème mathématique dans un système de preuves. Des assistants automatiques fondés sur des règles de déduction sont ensuite utilisés pour faire la preuve de ce théorème. Cependant, un opérateur humain doit guider ces assistants lorsqu'ils ne parviennent pas à démontrer certains lemmes. Ainsi, le processus global n'est pas totalement automatisé.

**Vérification de modèles.** Dans cette dernière approche, l'ensemble de la procédure est automatique et permet de garantir que le modèle du système vérifie une spécification. Celle-ci est représentée dans un formalisme logique et la démonstration de la propriété de correction est obtenue à l'aide d'algorithmes. Ceux-ci peuvent par exemple chercher à explorer de façon symbolique l'ensemble des comportements du modèle de sorte à s'assurer qu'aucun ne viole la formule décrivant la spécification.

Toutes les approches présentées précédemment sont en fait complémentaires. La méthode traditionnelle du test est également importante car elle permet d'étudier le système réel. Mais elle n'est pas toujours possible, par exemple lorsque l'utilisation du système réel est trop coûteuse ou trop dangereuse. La bonne méthode de vérification réside donc dans le choix d'un compromis adapté au type de système étudié.

Enfin, soulignons que ces méthodes se répandent de plus en plus dans les milieux industriels. Des normes officielles exigent que certains systèmes soient vérifiés et les entreprises dont l'intérêt est d'éviter les défauts de conception ou de réalisation, investissent de plus en plus dans ce domaine. Citons quelques exemples nationaux bien connus. La RATP, pour le développement de la partie critique du métro Meteor, a utilisé la méthode B. De même, les parties critiques des logiciels développés par Airbus ou EDF font appel aux méthodes formelles lors des phases de développement.

## Vérification de modèles

Nos travaux s'inscrivent dans le cadre de la vérification automatique de modèles, approche que nous allons décrire plus précisément. Celle-ci présente deux grandes étapes, qui sont représentées sur la figure 1.

- (1) **Modélisation du système :** le système étudié peut être donné sous la forme d'un système réel (système physique, ou code logiciel) ou sous la forme d'une description de son comportement (un protocole par exemple). Il est traduit dans un formalisme mathématique adapté, donnant ainsi un *modèle du système*, disons  $M$ .
- (2) **Modélisation de la spécification :** la spécification du système est souvent donnée par un cahier des charges. Elle est aussi traduite dans un formalisme mathématique, habituellement une logique,

donnant ainsi une *formule de la spécification*, disons  $\varphi$ .

- (3) **Vérification** : des algorithmes de vérification de modèles sont ensuite utilisés pour déterminer si le modèle du système satisfait la formule exprimant la spécification, ce qui est noté  $M \models \varphi$ . Si ce n'est pas le cas, des incohérences dans les descriptions du système et des propriétés sont mises en évidence par l'algorithme. Sinon, nous obtenons la garantie que le modèle  $M$  satisfait les propriétés exprimées par  $\varphi$ .

Le facteur critique de la vérification de modèles est la difficulté des problèmes étudiés. Ainsi, pour des classes trop générales de modèles ou de propriétés, il n'existe pas d'algorithmes de vérification. Et, même pour des propriétés simples, le passage à l'échelle est souvent difficile puisque la taille des systèmes étudiés peut être très grande. La complexité des algorithmes de vérification dépend à la fois de la classe de modèles considérés et de la logique de spécification prise en compte. D'une manière générale, plus la classe de modèles et/ou la logique sont expressives, plus le problème est difficile et il est donc nécessaire de trouver le bon compromis entre expressivité et efficacité.

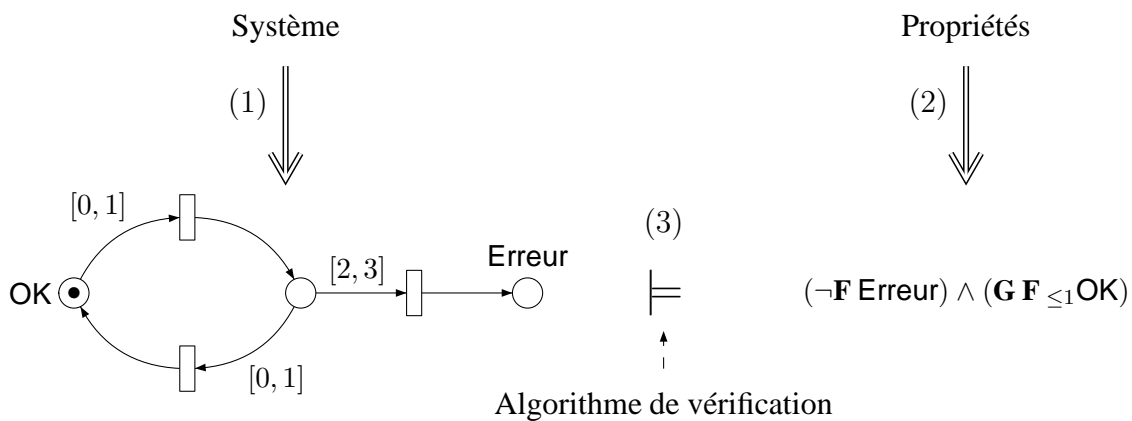


FIG. 1 – La vérification de modèles.

D'autre part, une faiblesse des méthodes formelles provient de la différence qui peut exister entre les modèles analysés et les systèmes réels. En effet, l'étape de modélisation qui consiste à plonger dans un cadre mathématique le système analysé, bien que nécessaire aux approches formelles, simplifie souvent le système. Il est donc naturel de s'intéresser à la conservation de ces propriétés lorsque le modèle est « concrétisé » afin d'obtenir un modèle plus proche d'un système réel. Le développement de techniques assurant l'implémentabilité des modèles, *i.e.* permettant un transfert des propriétés démontrées pour les modèles aux systèmes eux-mêmes, constitue un défi essentiel.

Nous nous intéressons dans la suite aux formalismes de description, à l'algorithmique de la vérification et à l'implémentabilité. Nous présentons plus précisément en quoi consistent ces trois domaines et les défis majeurs qu'ils présentent.

### Modélisation : Des formalismes de description

Lors de la phase de modélisation du système et des propriétés, un choix doit être fait parmi les nombreux formalismes de description existants. Les formalismes choisis doivent permettre d'exprimer les caractéristiques du système que nous souhaitons étudier. Ainsi, pour l'étude d'un protocole probabiliste, il semble naturel d'utiliser un modèle probabiliste comme les chaînes de Markov. Ces



formalismes doivent donc être suffisamment expressifs. Cependant, il est important de remarquer que le choix de ces formalismes a des conséquences sur l'ensemble du processus de vérification et d'implémentation. En effet, plus les modèles des systèmes et des propriétés sont expressifs, plus les algorithmes de vérification sont coûteux en temps et en espace. À l'opposé, plus le modèle du système est simple, moins il reflète les contraintes d'une implémentation, et donc plus l'implémentabilité est difficile.

Ces différents critères sont antagonistes et il est, en règle générale, assez délicat de choisir le bon modèle pour représenter le système et les propriétés. Les travaux portant sur les formalismes de description sont donc fondamentaux afin de permettre davantage de choix et afin d'offrir des modèles proposant un juste compromis.

Inventer de nouveaux modèles n'est pas difficile : il est relativement simple d'imaginer de nouvelles extensions d'un modèle existant. Cependant, une telle démarche, « vagabonde », n'est pas profitable. La conception d'un nouveau modèle doit être motivée et il existe pour cela différentes possibilités. Une étude de cas peut par exemple justifier l'ajout d'une structure de données à un modèle classique. D'autres motivations peuvent provenir de l'étude des modèles existants afin de mieux comprendre les propriétés qu'ils possèdent et celles qu'ils ne possèdent pas. Ceci permet ensuite de dégager des attentes concernant le nouveau modèle.

La comparaison des modèles existants constitue une autre facette de ces travaux. En effet, à quoi bon disposer de plusieurs formalismes sans connaître leurs pouvoirs expressifs relatifs ? Analyser avec précision les relations en termes d'expressivité entre deux formalismes permet, par exemple, de déterminer quels types de systèmes peuvent être exprimés dans un modèle mais pas dans l'autre. D'autre part, il peut être préférable pour certaines classes de systèmes d'utiliser un modèle s'il permet des descriptions plus concises. Enfin, le développement d'algorithmes de traductions d'un formalisme vers un autre permet l'application d'algorithmes d'une classe à des objets d'une autre classe.

## **Vérification : Des problèmes d'algorithmique**

L'étape de vérification semble plus simple puisqu'il suffit d'utiliser un outil implémentant un algorithme de vérification adapté aux classes de systèmes et de propriétés choisies. Cependant, il existe fréquemment, pour une même classe, différents algorithmes dont les comportements sont incomparables. Il faut donc maîtriser les bases des théories sous-jacentes pour se repérer parmi les outils de vérification automatique de modèles.

Le principal défi de l'algorithmique de la vérification réside bien sûr dans le développement de nouveaux algorithmes efficaces en pratique. Ceci peut être délicat car les modèles considérés sont complexes et l'analyse exhaustive de leurs comportements ne termine pas en général. Nous présentons quelques techniques utiles dans ce cadre.

Citons les représentations symboliques qui permettent de manipuler des éléments de façon ensembliste, permettant des calculs plus efficaces directement au niveau de ces ensembles. Ainsi la représentation d'ensembles de valuations d'horloges sous forme de zones, largement répandue dans les algorithmes de vérification de systèmes temporisés, permet-elle de parcourir des ensembles infinis (même non dénombrables) d'objets en un temps fini. Le développement de nouvelles représentations symboliques et/ou d'algorithmes fondés sur ces représentations constitue donc un enjeu important. Par ailleurs, selon la nature de la propriété étudiée, les techniques d'analyse du modèle ne doivent pas être les mêmes. Par exemple, si la propriété ne concerne qu'un aspect restreint du comportement du système, il est possible de simplifier ce système lors de son analyse en faisant abstraction des caractéristiques qui sont inutiles.

Enfin, les algorithmes peuvent également prendre en compte la nature du modèle étudié et ses spécificités. Ainsi, lorsqu'un système est représenté comme la composition d'un ensemble de systèmes plus petits, l'approche simpliste consistant à calculer explicitement le système global correspondant à cette composition mène à des algorithmes inefficaces. Cette difficulté, connue comme le problème de l'explosion du nombre d'états, peut être évitée en développant des algorithmes tenant compte de la description du système sous forme de composition.

### **Implémentation : Du modèle au système réel**

L'étape d'implémentation est d'une certaine façon le processus inverse de la modélisation. Au lieu de simplifier le système, il s'agit cette fois de l'enrichir, de le raffiner, afin d'obtenir un objet plus proche de la réalité des contraintes physiques associées à un système matériel. Cette « concrétisation » peut également être perçue comme un cadre particulier du problème de synthèse. Celui-ci consiste à construire un programme ou un algorithme à partir de sa spécification, le système construit est alors correct par construction.

Naturellement, il n'est pas souhaitable de devoir réaliser une seconde étape de validation sur ce modèle concret, et il faut donc pouvoir transférer les propriétés du modèle le plus abstrait au modèle le plus concret.

Nous pouvons distinguer essentiellement deux types d'approches. La première consiste à modéliser explicitement les différences existant entre le « monde réel » et les modèles mathématiques. L'étape de vérification tient alors compte de ces aspects supplémentaires, et vérifie directement l'implémentation du modèle. Celle-ci est en général plus difficile à vérifier car sa taille est plus grande. Une seconde approche consiste à introduire les imprécisions dues à l'exécution du modèle abstrait dans un environnement matériel à travers la sémantique du modèle abstrait. La taille des modèles est donc préservée mais les algorithmes classiques de vérification doivent être adaptés à la nouvelle sémantique.

### **Modèles étudiés dans la thèse**

Nous nous intéresserons plus particulièrement à deux caractéristiques des systèmes : l'aspect temporel et l'aspect distribué.

Lors de la modélisation de systèmes, l'aspect temps-réel est souvent présent. Plus précisément, le seul ordonnancement des événements peut ne pas être suffisant pour caractériser les comportements du système. Il devient alors nécessaire de quantifier l'écoulement de temps intervenu entre deux actions. Les *modèles temporisés* permettent de représenter ces systèmes.

La représentation d'un système apparaît fréquemment sous la forme d'une composition de différents systèmes de plus petite taille. Il est donc utile de disposer d'un mécanisme de composition dans le formalisme choisi pour représenter le système. Une telle représentation possède deux avantages. Elle est à la fois plus concise et plus facile à comprendre car elle correspond à un découpage naturel de différentes composantes du système global. En effet, décrire sans composition un système équivalent à un système défini à l'aide de compositions induit un coût exponentiel. Cette construction revient à définir une notion d'état global du système décrivant dans quel état se trouve chacun des sous-systèmes. Les *modèles distribués* permettent une représentation sous forme de composition.

## Des modèles temporisés ...

Les modèles reconnus dans la communauté scientifique pour la représentation de systèmes temporisés sont assez nombreux. Nous nous intéressons à trois d'entre eux, les automates temporisés, les réseaux de Petri temporels et les réseaux de Petri temporisés.

Dans les automates temporisés, le temps est représenté à l'aide de variables explicites (les horloges) qui sont ajoutées au modèle standard des automates finis. Ces horloges évoluent toutes à la même vitesse, et peuvent être comparées à des constantes lors du franchissement d'une transition. Il est également possible de remettre les horloges à zéro.

Dans les deux extensions temporisées des réseaux de Petri mentionnées précédemment, le temps est introduit de façon implicite. Ainsi, dans les réseaux de Petri temporels, un « âge » est associé à chaque transition et représente la durée d'activation de la transition. Cette transition peut alors être franchie si et seulement si cette durée appartient à un certain intervalle. Dans les réseaux de Petri temporisés, ce sont les jetons qui possèdent un « âge » qui est contraint par un intervalle lors du franchissement d'une transition.

Les langages acceptés par ces modèles sont des langages temporisés, *i.e.* des ensembles de mots temporisés. Un mot temporisé est une séquence d'actions indiquant pour chaque action la date à laquelle elle a lieu.

L'urgence permet d'introduire une certaine vivacité dans les systèmes temporisés, *i.e.* une notion de progrès évitant que le système puisse rester indéfiniment dans un état sans franchir de transitions. Dans les automates temporisés, l'urgence est introduite au travers des invariants qui sont associés de façon locale aux états de contrôle du système. Dans les réseaux de Petri temporels, l'urgence est associée aux transitions activées : l'écoulement du temps ne peut pas désactiver une telle transition. Enfin, les réseaux de Petri temporisés sont dépourvus d'urgence et l'utilisation de conditions d'acceptation permet de pallier ce manque en forçant par exemple à visiter un état infiniment souvent.

## ... et aussi distribués

Les réseaux de Petri possèdent intrinsèquement les notions d'état local et de synchronisation. Dans le cadre non temporisé, ils sont très largement utilisés dans des milieux divers pour la modélisation de systèmes distribués. Les applications couvrent les domaines des systèmes de production, de transports, de communication, *etc.* Les deux modèles temporisés obtenus comme des extensions des réseaux de Petri conservent ces bonnes propriétés.

En revanche, le modèle des automates finis semble a priori moins adapté à la modélisation de systèmes distribués. En effet, la notion d'état local n'a pas de sens dans un automate fini seul et il existe une notion de synchronisation qui permet de faire « communiquer » différents automates finis entre eux. Ceci permet donc de représenter différentes composantes du système comme des automates finis puis de décrire leurs interactions à l'aide de cette fonction de synchronisation. Pour les transitions qui ne sont pas synchronisées, les différents automates sont concurrents. Il est facile d'étendre cette notion au modèle des automates temporisés, obtenant ainsi la notion de *réseaux d'automates temporisés*.

## Impact de la combinaison des deux aspects

La combinaison des aspects temporisés et distribués induit un certain nombre de difficultés. La première concerne la notion de temps global. Si le système est modélisé par différentes composantes, celles-ci doivent-elles posséder la même horloge globale ? Nous supposons que cette hypothèse doit être satisfaite. Cette horloge globale rend plus difficile l'analyse du système puisqu'elle induit

une dépendance entre les différentes composantes, dont les comportements ne sont plus totalement concurrents comme c'est le cas dans le cadre non temporisé.

Un autre point rendu plus délicat par la combinaison des aspects temporisés et distribués est la notion d'interblocages. Celle-ci correspond à des situations dans lesquelles le système est bloqué, *i.e.* ne peut plus franchir de transitions. Ceci peut par exemple survenir à cause d'incompatibilités discrètes, lorsqu'un processus doit franchir une transition synchronisée mais que le processus avec lequel il doit se synchroniser ne se trouve pas dans le bon état. L'introduction de contraintes temporelles induit deux types de difficultés supplémentaires. D'abord, des incompatibilités temporelles peuvent survenir si les contraintes temporisées associées aux deux transitions devant être synchronisées ne sont pas compatibles. Ensuite, la notion d'urgence peut induire d'autres blocages temporels puisqu'il est possible qu'un processus puisse localement laisser  $d$  unités de temps s'écouler mais que pour un autre processus, une contrainte d'urgence impose que strictement moins de  $d$  unités de temps s'écoulent. Ceci nécessite donc lors de l'analyse du système de prendre en compte les contraintes d'urgence de chaque processus, réduisant à nouveau la concurrence.

## Contributions de la thèse

Nos contributions s'inscrivent dans le cadre de la vérification de systèmes temporisés et distribués. Les modèles étudiés sont ceux décrits précédemment, *i.e.* les (réseaux d')automates temporisés, les réseaux de Petri temporels et les réseaux de Petri temporisés. Nos travaux portent sur chacun des trois aspects décrits précédemment, l'étude des formalismes de modélisation, l'algorithmique de la vérification, et l'implémentabilité. Afin d'alléger nos propos, nous ne donnons pas de références bibliographiques dans cette partie. Les travaux existants seront décrits plus précisément dans les introductions des différents chapitres.

### Étude des formalismes de modélisation

Nous avons travaillé principalement dans deux voies : l'analyse des fondements théoriques des automates temporisés et la comparaison des différents modèles temporisés présentés précédemment.

**Analyse des fondements théoriques des automates temporisés.** Les fondements théoriques des automates temporisés, et donc des langages temporisés *réguliers* qu'ils reconnaissent, ont toujours suscité un intérêt important. En effet les langages réguliers standards (non temporisés) possèdent un certain nombre de caractérisations algébriques et logiques utiles pour leur analyse tandis que les langages temporisés ne possèdent pas de telles caractérisations. En particulier, leur principal défaut est qu'ils ne sont pas clos par complémentation. Ceci rend plus délicate une définition équivalente en termes de logiques car les logiques sont (presque) toujours closes par négation. Il est alors possible d'abandonner la négation, de se restreindre à une sous-classe close par complémentation ou encore de chercher à mieux comprendre le rôle de la complémentation.

Suivant cette troisième voie, nous étudions des questions comme « un automate temporisé est-il complémentable ? » ou « un automate temporisé est-il déterminisable ? ». Il a été démontré que ces deux problèmes sont indécidables, ainsi que d'autres problèmes liés à la minimisation du nombre d'horloges ou à l'opération de mélange (« shuffle »). Par ailleurs, contrairement au cadre non temporisé, les transitions silencieuses augmentent le pouvoir expressif des automates temporisés. Il est donc naturel d'espérer que la classe des langages temporisés acceptés par des automates temporisés avec

transitions silencieuses possède de meilleures propriétés. Nous nous intéressons donc aux problèmes précédents pour cette classe de langages temporisés.

Nous répondons d'abord à une question centrale issue de l'introduction des transitions silencieuses : « Est-il possible de déterminer si le langage d'un automate temporisé avec transitions silencieuses peut être accepté par un automate temporisé sans transitions silencieuses ? » Nous démontrons que ce problème est indécidable. Nous étendons ensuite chacun des résultats d'indécidabilité précédents au cadre des automates temporisés avec transitions silencieuses, démontrant ainsi que cette dernière classe ne possède pas de meilleures propriétés que la classe des automates temporisés. Soulignons que les transitions silencieuses rendent plus délicate l'analyse de l'automate. En effet, dans un automate temporisé, étant donné un mot temporisé, il existe un nombre fini d'exécutions acceptant ce mot, ce qui n'est plus vrai en présence de transitions silencieuses.

**Comparaison de modèles temporisés.** Notre objectif principal est la comparaison du modèle des automates temporisés avec les deux extensions des réseaux de Petri introduites précédemment, car celles-ci peuvent être mieux adaptées à la modélisation de systèmes distribués.

Dans un premier temps, nous nous sommes intéressés au modèle des réseaux de Petri temporels. Ce modèle constitue l'extension temporisée la plus utilisée et pour laquelle il existe un grand nombre de travaux. Naturellement, les liens entre ces deux modèles ont donc déjà été étudiés et ont permis de montrer que pour la bisimulation, le modèle des automates temporisés est strictement plus expressif tandis que pour l'équivalence de langages, les deux modèles sont aussi expressifs.

Par ailleurs, il existe de nombreuses extensions des automates temporisés. Les contraintes d'horloges dites diagonales permettent de comparer à une constante le temps qui s'est écoulé entre deux actions. Ce type de contraintes peut être utile pour exprimer des conditions d'ordonnancements de tâches. Une autre extension naturelle consiste à autoriser la mise à jour des horloges à des valeurs entières.

Nous proposons dans ces travaux une traduction quadratique des réseaux d'automates temporisés étendus avec des contraintes d'horloges diagonales et des mises à jour intégrales vers les réseaux de Petri temporels. Cette traduction est même linéaire dès lors que le modèle n'utilise que l'une ou l'autre des deux extensions. Notre traduction préserve les langages acceptés et peut donc être utilisée pour décider de propriétés d'accessibilité ou d'accessibilité répétée. Elle permet également de démontrer que les réseaux de Petri temporels sont exponentiellement plus concis que les automates temporisés.

Dans un second temps, nous étudions le modèle des réseaux de Petri temporisés. Il est souvent mentionné que les réseaux de Petri temporisés sont plus expressifs que les automates temporisés. Pourtant, les automates temporisés et les réseaux de Petri temporisés saufs étendus avec des arcs de lecture sont bisimilaires. Afin de clarifier les liens entre ces deux modèles, nous nous intéressons au modèle des réseaux de Petri temporisés étendus avec des arcs de lecture.

Nous montrons d'abord que les automates temporisés et les réseaux de Petri temporisés bornés avec arcs de lecture sont bisimilaires, étendant ainsi le résultat précédent. Nous montrons également que le problème de couverture est décidable pour les réseaux de Petri temporisés avec arcs de lecture.

Nous nous intéressons ensuite à des questions d'expressivité et cherchons à déterminer si les arcs de lecture accroissent le pouvoir d'expression du modèle et s'ils sont nécessaires pour exprimer les automates temporisés. Nous montrons que c'est le cas uniquement pour les comportements Zeno. Nous présentons des constructions permettant de s'affranchir de ces arcs dans le cas des mots finis et des mots infinis non Zeno. Puisque les comportements Zeno ne correspondent pas à des comportements physiques réels, ces résultats expriment que le pouvoir d'expression des réseaux de Petri temporisés

est « suffisant » dans le cas général. Cependant ces constructions sont très complexes et ne sont donc pas utilisables dans des mécanismes de traduction.

Encore dans le domaine de l'expressivité, nous étudions le rôle des mises à jour non déterministes. Nous obtenons à nouveau que les différences apparaissent via les comportements Zeno. Dans ce cas, il est nécessaire de raffiner la granularité du modèle pour s'affranchir des mises à jour non déterministes. L'équivalence obtenue avec les automates temporisés nous permet enfin de déduire des résultats nouveaux sur les automates temporisés.

## Algorithmique de la Vérification

Nos contributions à l'algorithmique de la vérification des systèmes temporisés distribués suivent deux objectifs. D'abord, nous proposons des algorithmes originaux pour des modèles d'automates temporisés étendus, contenant par exemple des gardes diagonales ou des mises à jour d'horloges intégrales. Ensuite, nous développons un algorithme de dépliage pour les réseaux d'automates temporisés.

**Algorithmes pour des modèles d'automates temporisés étendus.** La transformation linéaire (ou quadratique) que nous obtenons pour les réseaux d'automates temporisés étendus permet d'appliquer au réseau de Petri temporel construit les algorithmes développés pour cette classe. J'ai implémenté cette traduction dans un prototype permettant de transformer des automates temporisés donnés selon le format de l'outil de UPPAAL en des modèles respectant la syntaxe de l'outil TINA qui permet d'analyser les réseaux de Petri temporels.

Par ailleurs, nous proposons un algorithme dédié au modèle des automates temporisés étendus avec des contraintes diagonales. L'algorithme standard pour l'analyse des automates temporisés est fondé sur la technique d'analyse en avant à la volée et utilise la représentation symbolique des zones. Son caractère « en avant » est essentiel car il permet la manipulation de variables entières au cours de l'analyse.

En présence de contraintes diagonales, cet algorithme n'est plus correct. De plus, les automates temporisés avec gardes diagonales provoquant une erreur de l'algorithme sont très rares. Nous cherchons donc à développer un nouvel algorithme ayant le même comportement que l'algorithme classique sur les automates ne provoquant pas d'erreurs.

Pour assurer cette propriété, nous utilisons la méthodologie du raffinement successif fondé sur l'analyse de contre-exemples. Dans notre cadre, le raffinement consiste à retirer des contraintes diagonales du système. L'analyse des contre-exemples qui constitue la partie la plus délicate de notre approche cherche donc à détecter quelles gardes diagonales sont responsables des erreurs de l'algorithme. Cette partie est réalisée à l'aide d'une modification de la structure de données des matrices de différences bornées permettant de représenter les zones.

J'ai implémenté cet algorithme dans l'outil de vérification UPPAAL, permettant ainsi de valider son efficacité.

**Construction d'un dépliage temporisé de réseaux d'automates temporisés.** Peu de travaux proposent aujourd'hui des algorithmes d'analyse des réseaux d'automates temporisés fondés sur des techniques d'ordres partiels. L'algorithme d'analyse en avant à la volée qui est le plus utilisé se contente de calculer en le parcourant l'automate produit équivalent au réseau et ne tire donc pas profit du caractère distribué du modèle.

Nous proposons un algorithme permettant la construction d'un dépliage temporisé d'un réseau d'automates temporisés avec invariants et horloges partagés. Afin de résoudre les difficultés mention-

nées précédemment et introduites par la combinaison des aspects temporisés et distribués (notion de temps global, blocages temporels, *etc*), nous introduisons des arcs de lecture dans le dépliage que nous construisons. Ces arcs permettent à la fois de modéliser les tests réalisés sur les horloges et d'exprimer les dépendances vis-à-vis des invariants.

Nous attachons ensuite des zones aux événements de ce dépliage. Ces zones représentent les contraintes temporisées associées au franchissement de cet événement et permettent de décrire l'ensemble des valeurs d'horloges accessibles à l'issue de ce franchissement. Nous obtenons donc avec ce dépliage temporisé une représentation de toutes les configurations accessibles du réseau. Nous proposons de plus une méthode de calcul efficace de cet objet ne manipulant que des zones de dimension bornée. Cet aspect est important car l'algorithmique des zones est cubique dans leur dimension.

Enfin, nous nous intéressons à la caractérisation d'un préfixe fini de ce dépliage qui soit complet, *i.e.* qui représente toutes les configurations accessibles du réseau. Nous proposons deux solutions différentes à ce problème. Dans les deux cas, ces préfixes finis permettent de décider en temps linéaire (dans la taille du préfixe) l'accessibilité d'une configuration ainsi que la possibilité de tirer une transition.

## Implémentabilité

Les automates temporisés, comme les extensions temporisées des réseaux de Petri, sont gouvernés par des sémantiques théoriques et idéalisées tandis que leur implémentation sur des matériaux informatiques réels est contrôlée par des lois physiques. Ces deux types de contraintes ne sont pas toujours compatibles et les implémentations des automates temporisés que nous considérons peuvent être sensiblement différentes de leur sémantique théorique.

Récemment, une nouvelle sémantique a été proposée pour les automates temporisés, intitulée sémantique « AASAP ». Ce nouveau cadre a permis de démontrer une relation entre la satisfaction d'une propriété d'accessibilité vis-à-vis de cette sémantique et l'existence d'une implémentation du système respectant cette propriété. Ceci a même donné lieu au développement d'une plate-forme permettant une démarche automatique depuis la description du modèle à la génération de code « sûr ».

Nous nous plaçons dans le cadre de cette sémantique pour les automates temporisés. Nous proposons des algorithmes permettant de résoudre le problème de la vérification robuste, *i.e.* tenant compte de cette nouvelle sémantique, pour des spécifications plus générales que l'accessibilité comme des propriétés de Büchi ou des propriétés exprimées dans la logique LTL. Les algorithmes que nous proposons sont fondés sur une extension de la construction de l'automate des régions. Nos algorithmes demeurent dans la classe de complexité PSPACE et nous démontrons qu'ils sont optimaux. Nous développons également un algorithme PSPACE pour la vérification de propriétés temporisées simples, comme la propriété de temps de réponse borné. Cet algorithme est *ad-hoc*, mais il constitue une première étape vers la vérification d'un ensemble plus grand de propriétés temporisées.

## Plan de la thèse

L'ensemble de cette thèse est divisé en quatre parties.

**Partie I.** Cette première partie est constituée des deux premiers chapitres et est dédiée à la *présentation des modèles temporisés* sur lesquels sont fondés nos travaux.

Dans le **chapitre 1**, nous présentons le modèle des automates temporisés. Pour cela, nous donnons d'abord un ensemble de définitions nécessaires à la description de systèmes temporisés, comme par

exemple les systèmes de transitions temporisés. Nous introduisons de plus les notions de synchronisations, d'équivalences de langage, de bisimulation, *etc.* Enfin, nous définissons les diverses extensions des automates temporisés auxquelles nous nous sommes intéressés par la suite : transitions silencieuses, mises à jour intégrales et non déterministes, contraintes diagonales et réseaux d'automates temporisés.

Dans le **chapitre 2**, nous présentons le modèle (non temporisé) des réseaux de Petri, puis deux extensions temporisées de celui-ci, les réseaux de Petri temporels et les réseaux de Petri temporisés.

**Partie II.** Dans la seconde partie, nous présentons l'ensemble de nos résultats entrant dans le cadre de *l'étude des formalismes de modélisation*.

Dans le **chapitre 3**, nous étudions les automates temporisés avec transitions silencieuses et démontrons plusieurs résultats d'indécidabilité pour ce modèle.

Dans le **chapitre 4**, nous nous intéressons aux liens existants entre les automates temporisés et les réseaux de Petri temporels. Plus précisément, nous présentons une traduction des automates temporisés étendus avec mises à jour intégrales et contraintes diagonales vers les réseaux de Petri temporels.

Dans le **chapitre 5**, nous tournons notre attention vers le modèle des réseaux de Petri temporisés. Nous étudions l'extension de ce modèle obtenue en ajoutant des arcs de lecture, démontrons la décidabilité du problème de couverture pour ce modèle, et l'équivalence entre les réseaux saufs de cette classe et les automates temporisés. Nous présentons enfin une série de résultats d'expressivité.

**Partie III.** La troisième partie aborde le domaine de *l'algorithmique de la vérification*.

Dans le **chapitre 6**, nous proposons un algorithme original pour décider l'accessibilité des automates temporisés avec contraintes diagonales. Nous donnons une présentation détaillée de l'algorithme classique d'analyse à la volé en avant sur lequel notre algorithme repose. Nous expliquons pourquoi il n'est pas correct en présence de gardes diagonales et, après avoir décrit les différentes possibilités pour le corriger, nous présentons notre approche fondée sur le raffinement successif guidé par l'analyse des contre-exemples.

Dans le **chapitre 7**, nous nous intéressons à la définition d'un algorithme de dépliage pour le modèle des réseaux d'automates temporisés avec invariants et horloges partagées. Nous présentons d'abord la construction classique d'un dépliage pour les réseaux d'automates finis puis nous détaillons les différentes étapes de nos travaux permettant de proposer un algorithme efficace pour la construction d'un préfixe fini du dépliage temporisé d'un réseau d'automates temporisés.

**Partie IV.** La quatrième partie enfin concerne la problématique de *l'implémentation du système*.

Cette dernière partie est composée du **chapitre 8**. Celui-ci commence par présenter le cadre de la sémantique AASAP et énonce ses principales propriétés. Nous présentons ensuite nos algorithmes pour la vérification de propriétés co-Büchi, de formules LTL et enfin de certaines propriétés temporisées.

Nous donnons enfin une conclusion générale à l'ensemble de ces travaux, en décrivant plusieurs grandes perspectives.



**Première partie**

**Modèles Temporisés**



# Chapitre 1

## Automates temporisés

Dans ce premier chapitre, nous introduisons le modèle le plus couramment utilisé pour l'étude et l'analyse des systèmes temporisés, le modèle des automates temporisés. Nous commençons par introduire quelques notions et définitions préliminaires (section 1.1), puis nous présentons dans la section 1.2 un modèle plus général permettant de définir la sémantique des automates temporisés, les systèmes de transitions temporisés. Nous définissons alors dans la section 1.3 le modèle classique des automates temporisés, tel qu'il a été introduit par Alur et Dill dans [AD90]. Nous donnons ensuite les résultats fondamentaux de décidabilité de ce modèle (section 1.4) et nous présentons dans la section 1.5 des extensions de ce modèle, que nous serons amenés à utiliser dans la suite de ce travail. Dans la section 1.6, nous présentons enfin son extension aux systèmes distribués, les réseaux d'automates temporisés.

### 1.1 Préliminaires

Dans cette première section, nous introduisons quelques notions fondamentales pour l'étude des systèmes temporisés. En effet, afin de prendre en compte les notions quantitatives de temps, nous devons utiliser des objets mathématiques adaptés.

#### 1.1.1 Quelques éléments de mathématiques

Nous notons  $\mathbb{R}$  (respectivement  $\mathbb{R}_{\geq 0}$ ,  $\mathbb{R}_{> 0}$ ) l'ensemble des nombres réels (respectivement réels positifs, réels strictement positifs). De même, nous définissons les ensembles  $\mathbb{Q}$ ,  $\mathbb{Q}_{> 0}$  et  $\mathbb{Q}_{\geq 0}$ . Nous notons également  $\mathbb{N}$  l'ensemble des entiers naturels,  $\mathbb{Z}$  l'ensemble des entiers relatifs et  $\mathbb{N}_{> 0}$  l'ensemble des entiers naturels privé de 0.

Étant donné un nombre réel  $x \in \mathbb{R}$ , nous notons  $\lfloor x \rfloor$  sa partie entière, définie par  $\lfloor x \rfloor = \max\{i \in \mathbb{Z} \mid i \leq x\}$ . Sa partie fractionnaire est notée  $\langle x \rangle$  et définie par  $\langle x \rangle = x - \lfloor x \rfloor$ . De plus, nous étendons l'opérateur *modulo* aux nombres rationnels strictement positifs. Soit  $r \in \mathbb{Q}_{> 0}$ , alors étant donné un nombre réel  $x \in \mathbb{R}$ , nous définissons le reste de  $x$  modulo  $r$ , noté  $x \bmod r$ , par  $x \bmod r = x - \lfloor \frac{x}{r} \rfloor r$ . Nous écrivons enfin  $x \equiv y \bmod r$  si et seulement si  $(x - y) \bmod r = 0$ .

Étant donnée une fonction  $f$  définie de  $X$  dans  $Y$ , nous étendons naturellement  $f$  à l'ensemble des sous-ensembles de  $X$ , par  $f(X') = \{f(x) \mid x \in X'\}$ , pour tout sous-ensemble  $X'$  de  $X$ . Nous définissons également l'image réciproque  $f^{-1}$  sur l'ensemble des parties de  $Y$ , par  $f^{-1}(Y') = \{x \in X \mid f(x) \in Y'\}$  pour tout sous-ensemble  $Y'$  de  $Y$ .

### 1.1.2 Alphabet, mots temporisés et mots de délais

**Cadre non temporisé.** Notons  $\Sigma$  un ensemble fini, que nous appelons *alphabet*. Les éléments de  $\Sigma$  sont appelés les *actions*. L'ensemble des séquences finies (respectivement infinies) d'éléments de  $\Sigma$ , appelées *mots finis sur  $\Sigma$*  (respectivement *mots infinis sur  $\Sigma$* ), est noté  $\Sigma^*$  (respectivement  $\Sigma^\omega$ ). Nous noterons  $\Sigma^\infty$  l'union  $\Sigma^* \cup \Sigma^\omega$  de ces deux ensembles. Enfin, nous définissons la longueur d'un mot  $w \in \Sigma^\infty$ , notée  $|w|$ , par :

$$|w| = \begin{cases} n & \text{si } w = w_1 \dots w_n \in \Sigma^* \\ +\infty & \text{si } w \in \Sigma^\omega \end{cases}$$

De même, étant donné  $a \in \Sigma$  et  $w \in \Sigma^\infty$ , nous définissons la longueur de  $w$  en  $a$  comme le nombre d'occurrences de  $a$  dans  $w$ , noté  $|w|_a$  et défini par  $|w|_a = \text{Card}(\{1 \leq i \leq |w| \mid w_i = a\})$ , où  $\text{Card}$  dénote la fonction qui donne le cardinal d'un ensemble.

**Mots temporisés.** Nous considérons comme *domaine de temps*  $\mathbb{T}$  l'ensemble  $\mathbb{Q}_{\geq 0}$  des nombres rationnels positifs ou l'ensemble  $\mathbb{R}_{\geq 0}$  des nombres réels positifs. Dans toute la suite,  $\mathbb{T}$  représente donc indifféremment  $\mathbb{Q}_{\geq 0}$  ou  $\mathbb{R}_{\geq 0}$ . Une *séquence de dates* sur  $\mathbb{T}$  est une séquence  $\tau = (\tau_i)_{1 \leq i}$ , vérifiant pour tout  $i \geq 1$ ,  $\tau_i \in \mathbb{T}$  et  $\tau_i \leq \tau_{i+1}$ . Cette séquence peut être finie ou infinie. Un *mot temporisé fini*  $w = (a_i, \tau_i)_{1 \leq i \leq p}$  est un élément de  $(\Sigma \times \mathbb{T})^*$ , où  $\sigma = (a_i)_{1 \leq i \leq p}$  est un mot fini de  $\Sigma^*$  et  $\tau = (\tau_i)_{1 \leq i \leq p}$  une séquence finie de dates sur  $\mathbb{T}$  de même longueur. La date  $\tau_i$  correspond intuitivement à l'instant auquel l'action  $a_i$  a lieu. Nous définissons de façon similaire les mots temporisés infinis. L'application naturelle qui à un mot temporisé (élément de  $(\Sigma \times \mathbb{T})^\infty$ ) associe le mot (non temporisé) sur  $\Sigma$  (élément de  $\Sigma^\infty$ ) obtenu en effaçant les dates est notée  $\text{NON-TEMP}$ . Étant donné un mot temporisé  $w = (a_i, \tau_i)_{i \geq 1} \in \mathcal{MT}(\Sigma)$  et une lettre  $a \in \Sigma$ , nous définissons la longueur de  $w$  en  $a$ , notée  $|w|_a$ , par la longueur  $|\sigma|_a$  où  $\sigma = \text{NON-TEMP}(w)$ .

L'ensemble des mots temporisés finis (respectivement infinis) sur  $\Sigma$  est noté  $\mathcal{MT}^*(\Sigma)$  (respectivement  $\mathcal{MT}^\omega(\Sigma)$ ). L'union de ces deux ensembles est notée  $\mathcal{MT}(\Sigma)$ . Un sous-ensemble de  $\mathcal{MT}(\Sigma)$  est appelé un *langage temporisé*.

**Mots de délais.** Il existe une seconde représentation des mots temporisés qui lui est équivalente. Son principe consiste à indiquer les délais intermédiaires entre les actions, au lieu de leurs dates. Un *mot de délais fini*  $\delta = (d_1, a_1).(d_2, a_2) \dots (d_n, a_n)$  sur  $\Sigma$  est un élément de  $(\mathbb{T} \times \Sigma)^*$ . La durée  $d_1$  représente la date à laquelle l'action  $a_1$  a lieu. La durée  $d_i$ , pour  $i \geq 1$ , représente le temps écoulé entre les deux actions  $a_{i-1}$  et  $a_i$ . De façon similaire, nous définissons un *mot de délais infini* comme un élément de  $(\mathbb{T} \times \Sigma)^\omega$ . Nous définissons enfin, en utilisant les notations précédentes, la bijection *Délai* qui permet d'associer à un mot temporisé  $w = (a_i, \tau_i)_{1 \leq i \leq p}$  un mot de délais  $\delta = (d_i, a_i)_{1 \leq i \leq p}$  de même longueur ( $p$  peut être égal à  $+\infty$ ) défini par :

$$\text{Délai}(w) = \delta \text{ avec } d_i = \begin{cases} \tau_i & \text{si } i = 1 \\ \tau_i - \tau_{i-1} & \text{si } 1 < i \leq p. \end{cases}$$

**Action silencieuse.** Un élément supplémentaire peut être ajouté à l'alphabet des actions, l'*action silencieuse*  $\varepsilon$ . Cette action se comporte comme un élément neutre pour le monoïde  $\Sigma^*$  muni de la loi de composition interne définie par la concaténation, ce qui justifie son appellation. Nous supposons que l'élément  $\varepsilon$  n'appartient pas à l'alphabet d'actions  $\Sigma$  et nous noterons  $\Sigma_\varepsilon$  l'alphabet  $\Sigma \uplus \{\varepsilon\}$ , où  $\uplus$  désigne l'union ensembliste disjointe. Les définitions précédentes présentées pour l'alphabet  $\Sigma$  s'appliquent à l'alphabet  $\Sigma_\varepsilon$ . Nous obtenons ainsi les notations  $\mathcal{MT}^*(\Sigma_\varepsilon)$ ,  $\mathcal{MT}^\omega(\Sigma_\varepsilon)$  et  $\mathcal{MT}(\Sigma_\varepsilon)$ .

Nous notons  $\pi_\Sigma$  l'application qui associe à un mot temporisé (fini ou infini) sur  $\Sigma_\varepsilon$  sa projection sur  $\Sigma$ , obtenue en effaçant les actions étiquetées par  $\varepsilon$ . Il est important de remarquer qu'avec cette projection, un mot infini peut avoir pour image un mot fini. Ceci correspond à des exécutions infinies ne produisant qu'un nombre fini d'actions « visibles ». Dans la suite de ces travaux, nous ne considérerons pas ces comportements, puisqu'il semble raisonnable de supposer que si le nombre d'actions internes à un système est infini, alors le nombre d'actions visibles doit également l'être.

**Comportements Zeno.** Une autre classe de mots temporisés mérite d'être distinguée. Nous définissons, pour un mot temporisé (fini ou infini)  $w = (a_1, \tau_1) \dots (a_n, \tau_n) \dots$  la *durée* du mot  $w$ , notée  $Durée(w)$ , par  $Durée(w) = \sup_k \tau_k$ . Un mot temporisé infini  $w$  défini sur  $\Sigma$  est alors dit *Zeno* si sa durée  $Durée(w)$  est finie. Ceci correspond donc à des comportements infinis réalisés dans un temps fini. Du point de vue des applications, ces comportements doivent être exclus puisqu'aucune réalisation physique ne peut vérifier une telle condition. Nous définissons la sous-classe des *mots temporisés infinis non Zeno* et la notons  $\mathcal{MT}^{\omega_{nz}}(\Sigma)$ .

**Exemple 1.1** (Mots temporisés et mots de délais). Nous donnons quelques exemples de mots temporisés afin d'illustrer les classes introduites précédemment. Fixons un alphabet  $\Sigma = \{a, b\}$ . Dans les définitions suivantes, le mot  $w_1$  appartient à la classe  $\mathcal{MT}^*(\Sigma)$ , tandis que le mot  $w_2$  appartient à la classe  $\mathcal{MT}^*(\Sigma_\varepsilon)$  et vérifie  $\pi_\Sigma(w_2) = w_1$ . De plus, les deux mots temporisés infinis  $w_3$  et  $w_4$  sont éléments de  $\mathcal{MT}^\omega(\Sigma)$ , mais seul  $w_4$  est non Zeno, puisque la séquence de dates qui lui est associée (i.e.  $(i)_{i \geq 1}$ ) diverge vers  $+\infty$ .

$$\begin{aligned}
w_1 &= (a, 1)(b, 1)(b, 1.3)(a, 4.9) \\
w_2 &= (a, 1)(\varepsilon, 1)(b, 1)(b, 1.3)(\varepsilon, 4)(a, 4.9)(\varepsilon, 12) \\
w_3 &= (a, \tau_1) \dots (a, \tau_n) \dots \text{ avec } \tau_n = 1 - \frac{1}{2^n} \\
w_4 &= (a, 0)(b, 1) \dots (a, 2n)(b, 2n + 1) \dots \\
\text{Délai}(w_1) &= (a, 1).(b, 0).(b, 0.3).(a, 3.6) \\
\text{Délai}(w_2) &= (a, 1).(\varepsilon, 0).(b, 0).(b, 0.3).(\varepsilon, 2.7).(a, 0.9).(\varepsilon, 7.1) \\
\text{Délai}(w_3) &= (a, d_1) \dots (a, d_n) \dots \text{ avec } d_n = \frac{1}{2^n} \\
\text{Délai}(w_4) &= (a, 1).(b, 1) \dots (a, 1).(b, 1) \dots
\end{aligned}$$

┘

### 1.1.3 Horloges, valuations, remises à zéro et contraintes d'horloges

**Horloges, valuations et remises à zéro.** Nous considérons un ensemble fini  $X$  de variables que nous appellerons *horloges*. Une *valuation d'horloges* sur  $X$  est une application  $v : X \rightarrow \mathbb{T}$  qui associe à chaque horloge une valeur de temps. L'ensemble des valuations d'horloges sur  $X$  est noté  $\mathbb{T}^X$ . Nous décrivons plusieurs opérations sur ces valuations. La première représente l'écoulement du temps. Étant donnée une valeur  $t \in \mathbb{T}$  et une valuation  $v \in \mathbb{T}^X$ , nous définissons la valuation  $v + t$  par  $(v + t)(x) = v(x) + t, \forall x \in X$ . La seconde opération consiste à remettre à zéro les valeurs d'un certain sous-ensemble d'horloges. Pour un sous-ensemble  $R$  de  $X$ , nous notons  $v[R \leftarrow 0]$  la valuation définie ainsi :

$$(v[R \leftarrow 0])(x) = \begin{cases} 0 & \text{si } x \in R, \\ v(x) & \text{si } x \in X \setminus R. \end{cases}$$

L'opération consistant à remettre à zéro l'horloge  $x$  est notée  $x := 0$ . Enfin, nous définissons la valuation  $\mathbf{0}$  par  $\mathbf{0}(x) = 0$  pour toute horloge  $x \in X$ .

**Contraintes d'horloges.** Étant donné un ensemble d'horloges  $X$ , nous définissons l'ensemble des contraintes d'horloges sur  $X$ , noté  $\mathcal{C}(X)$  et engendré par la grammaire suivante :

$$\varphi ::= x \sim c \mid x - y \sim c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{tt}$$

où  $x, y \in X, c \in \mathbb{Q}, \sim \in \{<, \leq, =, \geq, >\}$ . Dans cette définition et dans la suite de ce document, le symbole  $\mathbf{tt}$  (qui vient de l'anglais « true ») dénote le booléen « vrai ». Nous utilisons de même la notation  $\mathbf{ff}$  pour le booléen « faux ».

Nous appelons *contraintes non-diagonales* les contraintes comparant une horloge à une constante, i.e. de la forme  $x \sim c$ , avec  $x \in X$  et  $c \in \mathbb{Q}$ . Au contraire, les contraintes comparant à une constante la différence entre deux horloges (de la forme  $x - y \sim c$  avec  $x, y \in X$  et  $c \in \mathbb{Q}$ ) sont appelées *contraintes diagonales*. L'ensemble des contraintes d'horloges non-diagonales, noté  $\mathcal{C}_{nd}(X)$ , est engendré par la grammaire :

$$\varphi ::= x \sim c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{tt}$$

où  $x \in X, c \in \mathbb{Q}, \sim \in \{<, \leq, =, \geq, >\}$ .

Nous distinguons enfin une seconde sous-classe propre de contraintes, l'ensemble des contraintes de borne supérieure, noté  $\mathcal{C}_{bs}(X)$  et engendré par la grammaire suivante :

$$\varphi ::= x \sim c \mid \varphi \wedge \varphi \mid \mathbf{tt}$$

où  $x \in X, c \in \mathbb{Q}, \sim \in \{<, \leq\}$ .

Considérons à présent une contrainte d'horloge  $\varphi$  dans laquelle toutes les constantes sont entières. Nous disons que  $\varphi$  est *k-bornée*,  $k$  étant un entier naturel, si toutes les constantes qui apparaissent dans sa définition sont bornées (en valeur absolue) par  $k$ .

La relation de satisfaction  $\models$  pour les contraintes d'horloges est définie sur l'ensemble des valuations d'horloges de  $X$ . Cette définition procède par induction comme suit ( $v$  désigne une valuation de  $\mathbb{T}^X$ ) :

$$\begin{aligned} v \models x \sim c & \iff v(x) \sim c \\ v \models x - y \sim c & \iff v(x) - v(y) \sim c \\ v \models \varphi_1 \wedge \varphi_2 & \iff v \models \varphi_1 \text{ et } v \models \varphi_2 \\ v \models \varphi_1 \vee \varphi_2 & \iff v \models \varphi_1 \text{ ou } v \models \varphi_2 \\ v \models \mathbf{tt} & \iff \mathbf{tt} \end{aligned}$$

Si la relation  $v \models \varphi$  est vraie, nous dirons que la valuation  $v$  satisfait, ou vérifie, la contrainte  $\varphi$ . Nous définissons également l'ensemble des valuations d'horloges qui satisfont une contrainte  $\varphi$ , noté  $\llbracket \varphi \rrbracket$ , par  $\llbracket \varphi \rrbracket = \{v \in \mathbb{T}^X \mid v \models \varphi\}$ . Étant donnée une horloge  $x \in X$  et une contrainte d'horloge  $\varphi \in \mathcal{C}(X)$ , nous définissons la projection sur l'horloge  $x$  des valuations de  $\llbracket \varphi \rrbracket$ , noté  $\llbracket \varphi \rrbracket|_x$ , par  $\llbracket \varphi \rrbracket|_x = \{v(x) \mid v \in \llbracket \varphi \rrbracket\}$ .

Étant donnée une contrainte d'horloge  $\varphi \in \mathcal{C}(X)$ , nous définissons l'ensemble des horloges apparaissant dans  $\varphi$ , noté  $\text{Horloges}(\varphi)$ . Cette définition procède par induction sur  $\varphi$  de la manière suivante :

$$\begin{aligned} \text{Horloges}(x \sim c) & = \{x\} \\ \text{Horloges}(x - y \sim c) & = \{x, y\} \\ \text{Horloges}(\varphi_1 \wedge \varphi_2) & = \text{Horloges}(\varphi_1) \cup \text{Horloges}(\varphi_2) \\ \text{Horloges}(\varphi_1 \vee \varphi_2) & = \text{Horloges}(\varphi_1) \cup \text{Horloges}(\varphi_2) \\ \text{Horloges}(\mathbf{tt}) & = \emptyset \end{aligned}$$

### 1.1.4 Intervalles de temps

Il peut également être nécessaire d'exprimer des contraintes temporelles sans expliciter la variable correspondante. Nous utiliserons alors la notion d'*intervalles de temps*. Un intervalle  $I$  de  $\mathbb{R}_{\geq 0}$  est un  $\mathbb{Q}_{\geq 0}$ - (respectivement un  $\mathbb{N}$ -) *intervalle* si sa borne inférieure appartient à  $\mathbb{Q}_{\geq 0}$  (respectivement à  $\mathbb{N}$ ) et sa borne supérieure appartient à  $\mathbb{Q}_{\geq 0} \cup \{+\infty\}$  (respectivement  $\mathbb{N} \cup \{+\infty\}$ ). Nous notons  $\mathcal{I}$  (respectivement  $\mathcal{I}_{\mathbb{N}}$ ) l'ensemble des  $\mathbb{Q}_{\geq 0}$ - (respectivement  $\mathbb{N}$ -) intervalles de  $\mathbb{R}_{\geq 0}$ . Nous définissons la *fermeture par le bas* d'un intervalle  $I \in \mathcal{I}$ , notée  $I^\downarrow$ , comme l'intervalle  $\{x \in \mathbb{T} \mid \exists y \in I, x \leq y\}$ . La fermeture par le bas d'un élément de  $\mathcal{I}$  (respectivement  $\mathcal{I}_{\mathbb{N}}$ ) est un élément de  $\mathcal{I}$  (respectivement de  $\mathcal{I}_{\mathbb{N}}$ ).

## 1.2 Systèmes de transitions temporisés

Nous définissons dans cette section le modèle le plus général permettant de décrire un système présentant une information quantitative du temps.

### 1.2.1 Définitions

Nous définissons d'abord les systèmes de transitions, objets mathématiques généraux dans lesquels le temps n'est pas encore représenté.

**Définition 1.2** (Système de Transitions). *Étant donné un alphabet  $\Sigma$  non nécessairement fini, un système de transitions est un triplet  $S = (Q, q_0, \rightarrow)$ , où  $Q$  est l'ensemble des états,  $q_0 \in Q$  est l'état initial et la relation de transition  $\rightarrow$  est un sous-ensemble de  $Q \times \Sigma \times Q$ . Une transition  $t = (q, a, q')$  sera représentée simplement par  $q \xrightarrow{a} q'$ .*

*Étant donné un système de transitions  $S = (Q, q_0, \rightarrow)$  et un sous-ensemble  $I \subseteq Q$  des états de  $S$ , nous définissons l'ensemble des états accessibles dans  $S$  depuis  $I$ , noté  $\text{Acc}(S, I)$ , et l'ensemble des états accessibles dans  $S$ , noté  $\text{Acc}(S)$ , par :*

$$\begin{aligned} \text{Acc}(S, I) &= \{q \in Q \mid \exists i \in I, i \rightarrow^* q\} \\ \text{Acc}(S) &= \text{Acc}(S, \{q_0\}) \end{aligned}$$

où  $\rightarrow^*$  représente la fermeture transitive de la relation  $\rightarrow$ .

**Comportements d'un système de transitions.** Soit  $S = (Q, q_0, \rightarrow)$  un système de transitions. Une exécution de ce système est une séquence  $\rho = q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \dots q_n \xrightarrow{a_n} \dots$  telle que pour tout indice  $i \geq 1$ , il existe une transition  $t_i = (q_i, a_i, q_{i+1})$  dans  $S$ . Le mot associé à une telle exécution est le mot  $w = (a_i)_{i \geq 1}$  et est appelé son étiquette.

**Langage accepté par un système de transitions.** Étant donné un système de transitions  $S = (Q, q_0, \rightarrow)$ , nous considérons d'une part un sous-ensemble de  $Q$ , appelé *ensemble d'états finals*, noté  $Q_f$ , et d'autre part un ensemble de sous-ensembles de  $Q$ , noté  $\{Q_r^1, \dots, Q_r^p\}$ . Ce second ensemble constitue une *condition de Büchi généralisée* pour  $S$ . Si cet ensemble est réduit à un singleton, nous dirons qu'il s'agit d'une *condition de Büchi*. De plus, chaque ensemble  $Q_r^i$  est appelé un *ensemble d'états répétés*.

Un mot fini  $w \in \Sigma^*$  est *accepté* par  $S$  si et seulement s'il existe une exécution finie de  $S$  issue de l'état initial, d'étiquette  $w$ , et menant à un état final. L'ensemble des mots finis acceptés par  $S$  est le *langage de mots finis* de  $S$  et est noté  $\mathcal{L}^*(S)$ .

Un mot infini  $w \in \Sigma^\omega$  est *accepté* par  $S$  si et seulement s'il existe une exécution infinie de  $S$  issue de l'état initial, d'étiquette  $w$ , et visitant infiniment souvent chacun des ensembles d'états répétés. L'ensemble des mots infinis acceptés par  $S$  est le *langage de mots finis* de  $S$  et est noté  $\mathcal{L}^\omega(S)$ .

Le *langage de  $S$* , noté  $\mathcal{L}(S)$ , est l'union des deux ensembles  $\mathcal{L}^*(S)$  et  $\mathcal{L}^\omega(S)$ .

**Systèmes de transitions temporisés.** Nous définissons les systèmes de transitions temporisés comme une sous-classe des systèmes de transitions dans laquelle certaines transitions représentent l'écoulement quantitatif de temps. Nous considérons un alphabet  $\Sigma$ .

**Définition 1.3** (Système de Transitions Temporisé). *Un système de transitions temporisé est un système de transitions  $S = (Q, q_0, \rightarrow)$ , où  $Q$  est l'ensemble des états,  $q_0 \in Q$  est l'état initial et la relation de transition  $\rightarrow$  est composée de transitions de délai  $q \xrightarrow{d} q'$  (avec  $d \in \mathbb{T}$ ) et de transitions discrètes  $q \xrightarrow{a} q'$  (avec  $a \in \Sigma$ ). De plus, la relation de transition  $\rightarrow$  doit vérifier les propriétés suivantes :*

**Déterminisme temporel :** si  $q \xrightarrow{d} q'$  et  $q \xrightarrow{d} q''$  avec  $d \in \mathbb{T}$ , alors  $q' = q''$  ;

**0-délai :**  $q \xrightarrow{0} q$  ;

**Additivité :** si  $q \xrightarrow{d} q'$  et  $q' \xrightarrow{d'} q''$  avec  $d, d' \in \mathbb{T}$ , alors  $q \xrightarrow{d+d'} q''$  ;

**Continuité :** si  $q \xrightarrow{d} q'$ , alors pour tout  $d'$  et  $d''$  appartenant à  $\mathbb{T}$  tels que  $d = d' + d''$ , il existe  $q'' \in Q$  tel que  $q \xrightarrow{d'} q'' \xrightarrow{d''} q'$ .

Les définitions d'ensemble d'états accessibles pour les systèmes de transitions s'appliquent aux systèmes de transitions temporisés.

**Comportements d'un système de transitions temporisé.** Avec ces propriétés, une *exécution temporisée* d'un système de transitions temporisé  $S$  peut s'écrire sous la forme d'une séquence  $\rho = q_1 \xrightarrow{d_1} q'_1 \xrightarrow{a_1} q_2 \dots q_n \xrightarrow{d_n} q'_n \xrightarrow{a_n} q_{n+1} \dots$  dans laquelle les transitions de délai alternent avec les transitions discrètes. Nous exigeons naturellement que pour tout indice  $i \geq 1$ , les transitions  $(q_i, d_i, q'_i)$  et  $(q'_i, a_i, q_{i+1})$  appartiennent à  $S$ . À une telle exécution est associé le mot de délais  $\delta = (a_i, d_i)_{1 \leq i \leq |w|}$  et le mot temporisé associé  $w = \text{Délai}^{-1}(\delta)$  (rappelons que Délai est une bijection). D'après la définition de Délai, en écrivant  $w = (a_i, \tau_i)_{1 \leq i \leq |w|}$ , nous obtenons  $\tau_i = \sum_{j=1}^i d_j$ . Nous disons alors que  $w$  est le *mot temporisé associé* à  $\rho$  et  $w$  est noté  $\text{Tempo}(\rho)$ . Il est aisé de remarquer que ces deux mots sont finis si et seulement si l'exécution est finie. Enfin, si l'alphabet de  $S$  est l'alphabet  $\Sigma_\varepsilon$ , i.e. qu'il contient l'action silencieuse  $\varepsilon$ , par projection  $(\pi_\Sigma)$  de  $w$  sur  $\Sigma$ , nous obtenons un mot temporisé  $w'$  sur  $\Sigma$ .  $\text{Tempo}(\rho)$  est alors défini par  $w'$ . De plus, le mot de délai associé à  $\rho$  dans ce cas est défini par  $\delta' = \text{Délai}(w')$ .

**Langage accepté par un système de transitions temporisé.** Comme pour les systèmes de transitions, nous associons à un tel système des conditions d'acceptation décrivant les exécutions acceptées. Celles-ci sont décrites par un ensemble d'états finals  $Q_f$  et une condition de Büchi généralisée  $\{Q_r^1, \dots, Q_r^p\}$ .

**Comportements finis.** Nous disons qu'une exécution temporisée finie  $\rho$  de  $S$  est *acceptante* si elle part de l'état initial  $q_0$  et se termine dans un état de  $Q_f$ . Étant donné un mot temporisé fini  $w \in \mathcal{MT}^*(\Sigma)$ , nous disons que ce mot est *accepté par  $S$*  si et seulement s'il existe une exécution



temporisée *finie* acceptante  $\rho$  de  $S$  dont le mot temporisé associé  $\text{Tempo}(\rho)$  est égal à  $w$ . L'ensemble des mots temporisés finis ainsi acceptés par  $S$  est appelé le *langage temporisé de mots finis reconnu par  $S$*  et est noté  $\mathcal{L}^*(S)$ . Soulignons que nous avons imposé ici que l'exécution soit finie. Ainsi, en présence de transitions silencieuses, seules les exécutions finies peuvent accepter des mots finis, ce qui exclut le cas dégénéré des exécutions infinies dont le mot temporisé associé est fini.

**Comportements infinis.** Nous disons qu'une exécution temporisée infinie  $\rho$  de  $S$  est *acceptante* si et seulement si elle est issue de l'état initial  $q_0$  et si elle visite infiniment souvent chacun des ensembles répétés  $Q_r^i$ . Étant donné un mot temporisé infini  $w \in \mathcal{MT}^\omega(\Sigma)$ , nous disons que ce mot est *accepté par  $S$*  si et seulement si il existe une exécution temporisée *infinie* acceptante  $\rho$  de  $S$  dont le mot temporisé associé  $\text{Tempo}(\rho)$  est égal à  $w$ . L'ensemble des mots temporisés infinis ainsi acceptés par  $S$  est appelé le *langage temporisé de mots infinis reconnu par  $S$*  et est noté  $\mathcal{L}^\omega(S)$ .

**Langage accepté.** Nous définissons enfin le *langage temporisé reconnu par  $S$*  comme l'union  $\mathcal{L}(S) = \mathcal{L}^*(S) \cup \mathcal{L}^\omega(S)$ . Une sous-classe que nous serons amenés à étudier est le sous-ensemble constitué des mots infinis non Zeno de  $\mathcal{L}(S)$ , noté  $\mathcal{L}^{\omega_{nz}}(S)$ . Les différentes relations sont rappelées ci-dessous :

$$\begin{aligned}\mathcal{L}^*(S) &= \mathcal{L}(S) \cap \mathcal{MT}^*(\Sigma) \\ \mathcal{L}^\omega(S) &= \mathcal{L}(S) \cap \mathcal{MT}^\omega(\Sigma) \\ \mathcal{L}^{\omega_{nz}}(S) &= \mathcal{L}(S) \cap \mathcal{MT}^{\omega_{nz}}(\Sigma)\end{aligned}$$

## 1.2.2 Composition

Il apparaît parfois naturel de définir un système comme la composition de différents sous-systèmes. De nombreuses compositions existent pour les systèmes de transitions temporisés. Nous choisissons ici de présenter la composition introduite par Arnold et Nivat [Arn94] et fondée sur la *synchronisation des actions* des différents systèmes de transitions temporisés.

**Définition 1.4** (Fonction de synchronisation). *Une fonction de synchronisation  $n$ -aire sur  $\Sigma$  est une fonction partielle sur  $(\Sigma \cup \{\perp\})^n$  à valeurs dans  $\Sigma$ .*

**Définition 1.5** (Synchronisation de systèmes de transitions temporisés). *Étant donnée une famille finie de  $n \in \mathbb{N}_{>0}$  systèmes de transitions temporisés  $(S_i)_{1 \leq i \leq n}$  sur un alphabet  $\Sigma$  et une fonction de synchronisation  $n$ -aire sur  $\Sigma$ , nous définissons la synchronisation de la famille  $(S_i)_{1 \leq i \leq n}$  vis-à-vis de  $f$  comme le système de transitions temporisé  $S = (Q, q_0, \rightarrow)$  défini, en écrivant  $S_i = (Q_i, q_0^i, \rightarrow_i)$  pour tout  $i$ , par :*

- $Q = \prod_{1 \leq i \leq n} Q_i$  est l'ensemble des états,
- $q_0 = (q_0^i)_{1 \leq i \leq n}$  est l'état initial et
- pour tout  $q = (q_i)_{1 \leq i \leq n}$ ,  $q' = (q'_i)_{1 \leq i \leq n} \in Q$ ,
  - **transitions discrètes** : pour tout  $a \in \Sigma$ ,  $q \xrightarrow{a} q'$  si et seulement si il existe un élément  $(a_i)_{1 \leq i \leq n} \in (\Sigma \cup \{\perp\})^n$  tel que :
    - $f((a_i)_{1 \leq i \leq n}) = a$ ,
    - pour tout indice  $i$  tel que  $a_i \neq \perp$ ,  $q_i \xrightarrow{a_i} q'_i$ ,
    - pour tout indice  $i$  tel que  $a_i = \perp$ ,  $q'_i = q_i$ .
  - **transitions de délai** : pour tout  $d \in \mathbb{T}$ ,  $q \xrightarrow{d} q'$  si et seulement si pour tout indice  $i$ ,  $q_i \xrightarrow{d} q'_i$ .

Différents éléments de cette définition méritent d'être soulignés. D'abord, la fonction de synchronisation n'intervient que lors des transitions discrètes et permet de décrire quelles sont les composantes impliquées dans la synchronisation. De plus, lors d'une telle transitions discrète, les composantes du produit associées au symbole  $\perp$  sont inactives. D'autre part, les transitions de délai se comportent comme des transitions fortement synchronisées, *i.e.* impliquant l'ensemble des composantes.

Remarquons enfin que la taille (nombre d'états, de transitions) du système synchronisé est exponentielle en la taille de la description de la famille de systèmes et de la fonction de synchronisation.

### 1.2.3 Comparaison

Nous allons par la suite comparer différents modèles temporisés. Afin de disposer d'outils mathématiques, nous définissons des notions permettant de comparer des systèmes de transitions temporisés et des ensembles de systèmes de transitions temporisés.

**Définition 1.6** (Équivalences de langages). *Nous définissons la relation d'équivalence  $\equiv$  (respectivement  $\equiv_*$ ,  $\equiv_\omega$ ,  $\equiv_{\omega_{nZ}}$ ) pour les mots temporisés, (respectivement mots temporisés finis, infinis, infinis non Zeno), définie pour tous  $S_1, S_2$  systèmes de transitions temporisés par :*

$$\begin{aligned} S_1 &\equiv S_2 &\iff \mathcal{L}(S_1) &= \mathcal{L}(S_2) \\ S_1 &\equiv_* S_2 &\iff \mathcal{L}^*(S_1) &= \mathcal{L}^*(S_2) \\ S_1 &\equiv_\omega S_2 &\iff \mathcal{L}^\omega(S_1) &= \mathcal{L}^\omega(S_2) \\ S_1 &\equiv_{\omega_{nZ}} S_2 &\iff \mathcal{L}^{\omega_{nZ}}(S_1) &= \mathcal{L}^{\omega_{nZ}}(S_2) \end{aligned}$$

Ces notions peuvent être utilisées pour comparer des classes de modèles, *i.e.* des ensembles de systèmes de transitions temporisés.

**Définition 1.7** (Expressivité en termes de langages temporisés). *Soit  $S_1$  et  $S_2$  deux ensembles de systèmes de transitions temporisés et  $\mathcal{E}$  une des quatre équivalences de langages précédentes. Nous définissons la relation  $S_1 \subseteq_{\mathcal{E}} S_2$ , signifiant que  $S_2$  est plus expressive que  $S_1$  du point de vue de l'équivalence  $\mathcal{E}$  par :*

$$S_1 \subseteq_{\mathcal{E}} S_2 \iff \forall S_1 \in S_1, \exists S_2 \in S_2 \text{ tel que } S_1 \equiv_{\mathcal{E}} S_2$$

*Pour chacune des équivalences  $\mathcal{E}$ , nous étendons également les relations  $\equiv_{\mathcal{E}}$  aux ensembles de systèmes de transitions temporisés et définissons l'inclusion stricte  $\subsetneq_{\mathcal{E}}$  par :*

$$\begin{aligned} S_1 \equiv_{\mathcal{E}} S_2 &\iff S_1 \subseteq_{\mathcal{E}} S_2 \wedge S_2 \subseteq_{\mathcal{E}} S_1 \\ S_1 \subsetneq_{\mathcal{E}} S_2 &\iff S_1 \subseteq_{\mathcal{E}} S_2 \wedge S_1 \not\equiv_{\mathcal{E}} S_2 \end{aligned}$$

Remarquons que ces relations d'équivalences ne concernent que les langages reconnus par les systèmes de transitions temporisés, *i.e.* le résultat de l'exécution des systèmes, mais pas leurs comportements, indépendamment des conditions d'acceptation. Les notions de simulation et de bisimulation permettent de comparer l'ensemble des comportements de deux systèmes de transitions temporisés.

**Définition 1.8** (Simulation, Bisimulation). *Soit  $S_1 = (Q_1, q_0^1, \rightarrow_1)$  et  $S_2 = (Q_2, q_0^2, \rightarrow_2)$  deux systèmes de transitions temporisés sur l'alphabet  $\Sigma$ . Une relation  $\mathcal{R} \subseteq Q_1 \times Q_2$  est une simulation de  $S_1$  par  $S_2$  si elle vérifie les conditions suivantes :*

$$- (q_0^1, q_0^2) \in \mathcal{R},$$

- pour tout  $(q_1, q_2) \in \mathcal{R}$ , pour tout  $\sigma \in \Sigma \cup \mathbb{T}$ , pour tout  $q'_1 \in Q_1$  tel que  $(q_1, \sigma, q'_1) \in \rightarrow_1$ , il existe un état  $q'_2 \in Q_2$  tel que  $(q_2, \sigma, q'_2) \in \rightarrow_2$  et  $(q_2, q'_2) \in \mathcal{R}$ .

Nous disons dans ce cas que  $S_2$  simule  $S_1$  et nous le notons  $S_1 \sqsubseteq S_2$ . Cette notation se justifie par le fait que tout comportement de  $S_1$  est un comportement de  $S_2$ .

Une relation  $\mathcal{R} \subseteq Q_1 \times Q_2$  est une bisimulation de  $S_1$  par  $S_2$  si c'est une simulation de  $S_1$  par  $S_2$  et si la relation réciproque  $\mathcal{R}^{-1}$  est une simulation de  $S_2$  par  $S_1$ . S'il existe une telle relation, nous disons que  $S_1$  et  $S_2$  sont bisimilaires.

Afin de prendre en compte le fait que les transitions silencieuses correspondent à des comportements internes que nous souhaitons abstraire lors de l'étude du système, nous présentons des notions de simulation et de bisimulation *faibles* moins restrictives. Pour les différencier, nous dirons que les notions de simulation et de bisimulation précédentes sont *fortes*.

Étant donné un système de transitions temporisé  $S = (Q, q_0, \rightarrow)$  défini sur  $\Sigma_\varepsilon$ , nous définissons le système de transitions temporisé  $S^\varepsilon = (Q^\varepsilon, q_0^\varepsilon, \rightarrow_\varepsilon)$  défini sur  $\Sigma$  par :

- $Q^\varepsilon = Q$ ,
- $q_0^\varepsilon = q_0$ ,
- Pour tous  $q, q' \in Q^\varepsilon$ , nous avons
  - $\forall d \in \mathbb{T}, q \xrightarrow{d}_\varepsilon q' \iff \exists \rho : q \rightarrow^* q' \text{ dans } S \text{ tel que } \text{Tempo}(\rho) = (\varepsilon, d)$ ,
  - $\forall a \in \Sigma, q \xrightarrow{a}_\varepsilon q' \iff \exists \rho : q \rightarrow^* q' \text{ dans } S \text{ tel que } \text{Tempo}(\rho) = (a, 0)$ .

**Définition 1.9** (Bisimulation faible). *Étant donnés deux systèmes de transitions temporisés  $S_1$  et  $S_2$ , nous disons que  $S_2$  simule faiblement  $S_1$  si  $S_2^\varepsilon$  simule  $S_1^\varepsilon$ . De même,  $S_1$  et  $S_2$  sont dits faiblement bisimilaires si  $S_1^\varepsilon$  et  $S_2^\varepsilon$  sont bisimilaires.*

Le lemme suivant montre que la notion de bisimulation (même faible) est plus forte que l'équivalence de langages.

**Lemme 1.10.** *Soient deux systèmes de transitions temporisés  $S$  et  $S'$  et une relation de simulation (éventuellement faible)  $\mathcal{R}$  de  $S$  par  $S'$ . Considérons des ensembles d'états finals  $Q_f$  et  $Q'_f$  de  $S$  et  $S'$  et des conditions de Büchi généralisées  $\{Q_r^1, \dots, Q_r^p\}$  et  $\{Q_r^{1'}, \dots, Q_r^{q'}\}$ . Nous supposons de plus que  $\mathcal{R}$  vérifie les conditions suivantes :*

- (i)  $\forall q \in Q_f, (q, q') \in \mathcal{R} \Rightarrow q' \in Q'_f$
- (ii)  $\exists \psi$  fonction surjective de  $\{1, \dots, p\}$  dans  $\{1, \dots, q\}$  t.q.  $\forall q \in Q_r^i, (q, q') \in \mathcal{R} \Rightarrow q' \in Q_r^{\psi(i)'}$

Nous avons alors  $\mathcal{L}(S) \subseteq \mathcal{L}(S')$ .

En particulier, si  $\mathcal{R}$  est une relation de bisimulation telle que  $\mathcal{R}$  et  $\mathcal{R}^{-1}$  vérifient les conditions précédentes, nous obtenons  $S \equiv S'$ .

### 1.3 Automates temporisés d'Alur et Dill

Nous pouvons maintenant présenter le modèle des automates temporisés, tel qu'il a été introduit par Alur et Dill, dans leurs travaux [AD90, AD94]. Essentiellement, les automates temporisés sont une extension des automates finis, obtenue en ajoutant un ensemble fini d'horloges, qui peuvent être testées et remises à zéro lors du franchissement d'une transition.

**Définition 1.11** (Automates temporisés). *Un automate temporisé sur  $\mathbb{T}$  est un 8-uplet  $\mathcal{A} = (\Sigma, X, L, T, Inv, \ell_0, L_f, L_r)$  où :*

- $\Sigma$  est un alphabet fini d'actions,
- $X$  est un ensemble fini d'horloges,
- $L$  est un ensemble fini d'états de contrôle,
- $T \subseteq L \times \mathcal{C}_{nd}(X) \times \Sigma \times 2^X \times L$  est un ensemble fini de transitions,
- $Inv \in \mathcal{C}_{bs}(X)^L$  est une application associant à chaque état de contrôle un invariant, donné par une contrainte de borne supérieure sur  $X$ ,
- $\ell_0 \in L$  est l'état de contrôle initial,
- $L_f \subseteq L$  est l'ensemble des états de contrôle finals et
- $L_r \subseteq L$  est l'ensemble des états de contrôle répétés.

Nous noterons AT la classe des automates temporisés d'Alur et Dill ainsi définie. Une transition d'un automate temporisé est un objet de la forme  $t = (\ell, g, a, R, \ell') \in L \times \mathcal{C}_{nd}(X) \times \Sigma \times 2^X \times L$ . Nous utiliserons fréquemment la notation  $t = \ell \xrightarrow{g,a,R} \ell'$  pour représenter une telle transition. Nous dirons que  $g$  est la *garde* de cette transition,  $a$  son *étiquette* et  $R$  sa *remise à zéro*. De manière informelle, la garde correspond à la condition sur la valuation d'horloges sous laquelle la transition peut être franchie, l'étiquette est le nom de l'action associée au franchissement de cette transition et qui va être retenue par le mot temporisé correspondant et enfin la remise à zéro est l'opération qui doit être effectuée sur la valuation d'horloges à l'issue du franchissement de la transition. Nous définissons le comportement d'un automate temporisé, *i.e.* sa sémantique, en termes de système de transitions temporisé.

**Définition 1.12** (Sémantique d'un automate temporisé). *La sémantique d'un automate temporisé  $\mathcal{A}$  défini comme ci-dessus est le système de transitions temporisé  $\llbracket \mathcal{A} \rrbracket = (Q, q_0, \rightarrow)$  :*

- $Q = L \times \mathbb{T}^X$  est l'ensemble des états,
- $q_0 = (\ell_0, \mathbf{0})$  est l'état initial,
- $\rightarrow$  est définie par :
  - **transitions de délai** :  $(\ell, v) \xrightarrow{d} (\ell, v + d)$  si  $d \in \mathbb{T}$  et  $v + d \models Inv(\ell)$  ;
  - **transitions d'action** :  $(\ell, v) \xrightarrow{a} (\ell', v')$  s'il existe une transition  $t = (\ell, g, a, R, \ell') \in T$  de  $\mathcal{A}$  telle que  $v \models g$  et telle que  $v'$  définie par  $v' = v[R \leftarrow 0]$  vérifie  $v' \models Inv(\ell')$ .

De plus l'ensemble des configurations finales de  $\llbracket \mathcal{A} \rrbracket$ , noté  $Q_f$ , est défini par :

$$Q_f = \{(\ell, v) \mid \ell \in L_f, v \in \mathbb{T}^X\}$$

Pour les mots infinis, la condition d'acceptation que nous considérons est la condition de Büchi associée à l'ensemble de configurations répétées  $Q_r$  défini par :

$$Q_r = \{(\ell, v) \mid \ell \in L_r, v \in \mathbb{T}^X\}$$

**Remarque 1.13** (Conditions de Büchi généralisées). Il est possible de définir dans les contexte des automates temporisés des conditions de Büchi généralisées pour l'acceptation des mots infinis. Il suffit pour cela de considérer un ensemble  $\{L_r^1, \dots, L_r^p\}$  d'états de contrôle répétés puis d'associer l'ensemble  $Q_r^i$  défini comme précédemment à chaque  $L_r^i$ . Cependant, il est facile de montrer que pour les automates temporisés, toute condition de Büchi généralisée peut être remplacée par une condition de Büchi équivalente.  $\lrcorner$

De la définition 1.12 et des notions introduites dans la section 1.2, découlent les notions d'*exécution temporisée* dans un automate temporisé, d'*exécution acceptante*, de *mot accepté* par un automate temporisé et enfin de *langage temporisé reconnu* par un automate temporisé. Plus précisément, étant

donné un automate temporisé  $\mathcal{A}$ , nous définissons ces notions pour  $\mathcal{A}$  comme étant celles définies pour  $\llbracket \mathcal{A} \rrbracket$ . Nous noterons par exemple  $\mathcal{L}(\mathcal{A})$  le langage  $\mathcal{L}(\llbracket \mathcal{A} \rrbracket)$ . De plus, nous utiliserons également la notion de *chemin* dans un automate temporisé, définie comme étant une séquence  $(t_i)_{1 \leq i}$  de transitions adjacentes  $t_i = (\ell_i, g_i, a_i, R_i, \ell_{i+1})$ . Enfin, les définitions d'ensembles d'états accessibles  $\text{Acc}(\mathcal{A})$  et  $\text{Acc}(\mathcal{A}, I)$  (avec  $I \subseteq Q$ ) s'étendent naturellement aux automates temporisés. Nous en rappelons ici les définitions et introduisons un nouvel opérateur, noté **D-ACC**, décrivant les états de contrôle accessibles depuis la configuration initiale :

$$\begin{aligned} \text{Acc}(\mathcal{A}, I) &= \{(\ell, v) \mid \exists(\ell', v') \in I, (\ell', v') \rightarrow^* (\ell, v) \text{ dans } \llbracket \mathcal{A} \rrbracket\} \\ \text{Acc}(\mathcal{A}) &= \text{Acc}(\mathcal{A}, \{(\ell_0, \mathbf{0})\}) \\ \text{D-Acc}(\mathcal{A}) &= \{\ell \mid \exists v, (\ell, v) \in \text{Acc}(\mathcal{A})\} \end{aligned}$$

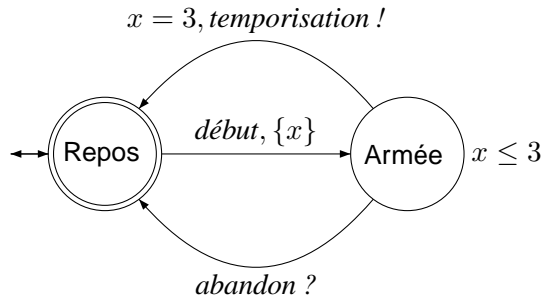


FIG. 1.1 – Un exemple d'automate temporisé  $\mathcal{A}_1$ .

**Exemple 1.14** (Un premier automate temporisé). Cet exemple présente l'automate temporisé  $\mathcal{A} = (\Sigma, X, L, T, \text{Inv}, \ell_0, L_f, L_r)$  avec :

- $\Sigma = \{\text{abandon ?}, \text{début}, \text{temporisation !}\}$ ,
- $X = \{x\}$ ,
- $L = \{\text{Repos}, \text{Armée}\}$ ,
- $T = \{(\text{Repos}, \text{tt}, \text{début}, \{x\}, \text{Armée}),$   
 $(\text{Armée}, x = 3, \text{temporisation !}, \emptyset, \text{Repos}),$   
 $(\text{Armée}, \text{tt}, \text{abandon ?}, \emptyset, \text{Repos})\}$ ,
- $\text{Inv}(\text{Repos}) = \text{tt}, \text{Inv}(\text{Armée}) = x \leq 3$ ,
- $\ell_0 = \text{Repos}$ ,
- $L_f = \{\text{Repos}\}$  et
- $L_r = \{\text{Repos}\}$ .

Cet automate modélise une minuterie simple. Celle-ci est au repos dans l'état de contrôle **Repos**, et est armée, *i.e.* déclenchée lorsqu'elle se trouve dans l'état de contrôle **Armée**. La transition étiquetée *début* remet à zéro l'horloge  $x$  et déplace l'état de contrôle vers l'état **Armée**. L'horloge  $x$  mesure alors la quantité de temps écoulée au sein de l'état **Armée**, et un invariant sur cet état impose que le système ne peut y rester plus de trois unités de temps. Pour le quitter, il y a deux possibilités, correspondant aux deux transitions sortantes. Ou bien la transition étiquetée *abandon ?* est franchie, qui symbolise qu'un message d'abandon a été reçu (le ? dans le nom de l'action représente une réception), ou bien le système atteint la limite de temps, et quitte l'état *via* la transition étiquetée par *temporisation !* (à nouveau, le symbole ! représente un envoi au système). Les mots finis sont constitués d'un nombre finis de tels cycles, tandis que les mots infinis sont constitués d'un nombre infini de tels cycles.

Une représentation graphique de cet automate est donnée sur la figure 1.1. Les états de l'automate sont représentés par des cercles, les transitions par des arcs reliant ces cercles. Les gardes, étiquettes et mises à jour des transitions sont inscrites à côté de celles-ci. Dans la suite, la remise à zéro de l'horloge  $x$  sera notée  $x := 0$  ou simplement  $\{x\}$ . De plus, les états initiaux (respectivement finals, répétés) sont indiqués par des flèches entrantes (respectivement des flèches sortantes, des doubles cercles). Enfin, les invariants sont indiqués à côté de l'état auquel ils correspondent. Lorsqu'une garde ou un invariant vaut tt, il n'est pas représenté. De même, si une remise à zéro est vide, elle n'est pas représentée. Le langage  $\mathcal{L}(\mathcal{A}_1)$  accepté par cet automate  $\mathcal{A}_1$  vérifie  $\mathcal{L}(\mathcal{A}_1) = \text{Délai}^{-1}(\mathcal{L}_1^* \cup \mathcal{L}_1^\omega)$  où  $\mathcal{L}_1$  est défini par

$$\mathcal{L}_1 = (\{\text{début}\} \times \mathbb{T}) \cdot [\{(\text{temporisation } !, 3)\} + (\{\text{abandon } ?\} \times [0, 3])]^*$$

$\mathcal{L}_1$  décrit l'ensemble des mots de délais associés aux mots temporisés acceptés lors d'un tour de boucle dans  $\mathcal{A}_1$ , *i.e.* lors d'une séquence *début temporisation !* ou *début abandon ?*.  $\lrcorner$

Nous introduisons enfin quelques définitions classiques sur les automates temporisés.

**Définition 1.15.** *Soit  $\mathcal{A}$  un automate temporisé.*

*Nous disons que  $\mathcal{A}$  est déterministe si, étant donné  $(\ell, g_1, a, R_1, \ell'_1)$  et  $(\ell, g_2, a, R_2, \ell'_2)$  deux transitions de  $\mathcal{A}$  issues du même état de contrôle  $\ell$  et étiquetées par la même action  $a$ , la contrainte  $g_1 \wedge g_2$  n'admet aucune solution.*

*Nous définissons la granularité de  $\mathcal{A}$  comme le plus grand multiple commun des fractions  $\frac{p}{q}$  intervenant dans la définition de  $\mathcal{A}$  (au niveau des gardes et des invariants). La granularité de  $\mathcal{A}$  est un élément de  $\mathbb{N}_{>0}$ .*

*Un sous-ensemble de  $\mathcal{MT}(\Sigma)$  est un langage temporisé régulier si et seulement s'il est accepté par un automate temporisé.*

## 1.4 Problème d'accessibilité, problème du vide

Dans cette section, nous nous intéressons au problème du vide pour les automates temporisés d'Alur et Dill. Dans un premier temps, nous définissons ce problème, puis nous donnons les grandes lignes de la preuve de sa décidabilité.

### 1.4.1 Pourquoi ces problèmes sont équivalents et fondamentaux

Le problème d'accessibilité d'un état de contrôle consiste à déterminer, étant donné un automate temporisé  $\mathcal{A}$  et un état de contrôle cible  $\ell$ , si cet état est accessible, *i.e.* s'il existe une exécution de  $\mathcal{A}$  qui atteint une configuration dont l'état de contrôle est  $\ell$ .

Le problème du vide consiste à déterminer si le langage accepté par un automate temporisé est vide ou non. Ces deux problèmes sont en fait équivalents. Pour le vérifier, il suffit de constater que les états de contrôle finals et cibles jouent en fait un rôle similaire dans chacun des deux problèmes.

Ces deux problèmes sont tout à fait fondamentaux et constituent la brique de base de la vérification. En effet, de nombreuses applications nécessitent de décider l'accessibilité. Ainsi, pour s'assurer qu'un système est sauf, on souhaite vérifier qu'il évite toujours les états « non-saufs », *i.e.* que ces derniers ne sont pas accessibles. Une propriété d'exclusion mutuelle sera typiquement exprimée dans ces termes.

### 1.4.2 Décidabilité des automates temporisés

Dans la suite, pour une classe<sup>1</sup> de modèles, nous dirons que cette classe est décidable si le problème du vide est décidable pour cette classe. Le résultat fondamental démontré par Alur et Dill dans les travaux fondateurs [AD90, AD94] justifie l'intérêt important porté à ce modèle dans le domaine de la vérification de systèmes temporisés :

**Théorème 1.16** ([AD90, AD94]). *La classe AT des automates temporisés d'Alur et Dill est décidable.*

Parce qu'elle est fondamentale et parce qu'elle sera réutilisée dans la suite de nos travaux, nous donnons ici les grandes lignes de la preuve de ce résultat. Étant donné un automate temporisé  $\mathcal{A}$ , l'idée de la preuve consiste à construire un nouvel automate fini  $\mathcal{B}$  tel que :

$$\mathcal{L}(\mathcal{B}) = \{\text{NON-TEMP}(w) \mid w \in \mathcal{L}(\mathcal{A})\}$$

où NON-TEMP, défini dans la section 1.1, décrit l'opération de projection des mots temporisés sur l'alphabet  $\Sigma$ . Nous avons alors la propriété suivante :

$$\mathcal{L}(\mathcal{B}) = \emptyset \iff \mathcal{L}(\mathcal{A}) = \emptyset$$

Comme le test du vide est décidable pour les automates finis ou de Büchi [HU79], il découle de cette équivalence que le test du vide est également décidable pour les automates temporisés. Dans la sous-section 1.4.3 nous allons détailler la construction de cet automate  $\mathcal{B}$ . Celle-ci requiert quelques simplifications que nous exposons maintenant.

**Invariants.** Une première étape consiste à supprimer les invariants de l'automate temporisé, *i.e.* à supposer que ces invariants sont tous égaux à tt. Étant donné un automate temporisé  $\mathcal{A}$ , nous considérons l'automate  $\mathcal{A}'$ , obtenu à partir de  $\mathcal{A}$  en réalisant les trois opérations suivantes :

1. pour tout état  $\ell$ , et toute transition  $t = (\ell, g, a, R, \ell')$ , remplacer  $t$  par  $t' = (\ell, g \wedge \text{Inv}(\ell), a, R, \ell')$ ,
2. pour tout état  $\ell'$ , toute transition  $t = (\ell, g, a, R, \ell')$ , et toute horloge  $x \in X$ , si  $x \notin R$  et si  $g_x$  est la contrainte exprimée par  $\text{Inv}(\ell')$  sur  $X$ , remplacer  $t$  par  $t' = (\ell, g \wedge g_x, a, R, \ell')$ , sinon vérifier que la valeur 0 est admissible pour  $x$  dans  $\text{Inv}(\ell')$ ,
3. remplacer la fonction  $\text{Inv}$  par  $\text{Inv}'(\ell) = \text{tt}$  pour tout état  $\ell$ .

Il est facile de vérifier que les langages acceptés par  $\mathcal{A}$  et  $\mathcal{A}'$  sont égaux. En effet, la première modification requiert simplement que lors de la sortie dans un état de contrôle, l'invariant de cet état est satisfait. La seconde, plus complexe, exprime la même condition lors de l'entrée dans un état de contrôle. Par la suite, nous ignorerons donc la présence des invariants pour le test du vide d'un automate temporisé<sup>2</sup>. Plus généralement, nous parlerons d'*automates temporisés sans invariants* lorsque ceux-ci seront sans intérêt.

**Constantes entières.** Dans un deuxième temps, nous énonçons un lemme, que nous réutiliserons par la suite. Celui-ci établit que l'on peut supposer que la granularité de l'automate temporisé vaut 1, *i.e.* toutes les constantes de l'automate temporisé sont des constantes entières (au lieu d'être rationnelles) :

<sup>1</sup>Par classe de modèles, nous entendons une extension ou une restriction d'automates temporisés.

<sup>2</sup>Mais alors à quoi servent-ils ? Nous le verrons lors de la synchronisation d'automates temporisés, page 41.

**Lemme 1.17** ([AD94]). *Soit  $\mathcal{A}$  un automate temporisé, et  $\lambda \in \mathbb{Q}_{>0}$ . Nous définissons l'automate  $\lambda.\mathcal{A}$  comme l'automate obtenu à partir de  $\mathcal{A}$  en multipliant toutes les constantes apparaissant dans les gardes des transitions et des invariants par  $\lambda$ . Soit  $w = (a_0, \tau_0)(a_1, \tau_1) \dots (a_n, \tau_n) \dots$  un mot temporisé. Nous avons alors :*

$$w \in \mathcal{L}(\mathcal{A}) \iff \lambda.w \in \mathcal{L}(\lambda.\mathcal{A})$$

où  $\lambda.w = (a_0, \lambda\tau_0)(a_1, \lambda\tau_1) \dots (a_n, \lambda\tau_n) \dots$  (toutes les dates sont multipliées par  $\lambda$ ).

Ainsi, multiplier toutes les constantes apparaissant dans un automate temporisé par une valeur rationnelle positive ne change pas le résultat du test du vide. En particulier, nous pouvons les multiplier par la granularité de l'automate de sorte à n'obtenir que des constantes entières.

Le reste de la preuve consiste donc à construire l'automate  $\mathcal{B}$ , en supposant que  $\mathcal{A}$  est sans invariants et que sa granularité vaut 1.

### 1.4.3 Automate des régions

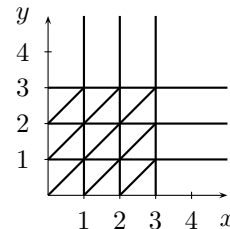
Soit  $\mathcal{A} = (\Sigma, X, L, T, Inv, \ell_0, L_f, L_r)$  un automate temporisé sans invariants et dont les constantes sont entières. Nous notons  $K$  la constante maximale en valeur absolue apparaissant dans  $\mathcal{A}$  (parmi les contraintes d'horloges). Nous allons définir sur l'espace (infini) des valuations d'horloges sur  $X$  une relation d'équivalence d'indice fini. Nous définissons la relation  $\approx$  ainsi : deux valuations  $v$  et  $v'$  sont équivalentes, noté  $v \approx v'$ , si et seulement si elles satisfont les conditions suivantes :

- pour toute horloge  $x \in X$ , ou bien  $v(x)$  et  $v'(x)$  sont strictement supérieures à  $K$ , ou bien  $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ ,
- pour toutes horloges  $x, x' \in X$ , si à la fois  $v(x)$  et  $v(x')$  sont inférieures à  $K$ , alors
  - $\langle v(x) \rangle \leq \langle v(x') \rangle$  ssi  $\langle v'(x) \rangle \leq \langle v'(x') \rangle$  et
  - $\langle v(x) \rangle = 0$  ssi  $\langle v'(x) \rangle = 0$ .

Ceci définit une relation d'équivalence, dont les classes d'équivalences sont appelées *régions*. Nous notons  $\mathcal{R}$  l'ensemble des régions,  $[v]$  la région contenant  $v$  et  $\bar{r}$  sa fermeture topologique pour la topologie habituelle de  $\mathbb{T}^X$ . Il est facile de vérifier que le nombre de régions est borné par  $n! \cdot 2^n \cdot (2K + 2)^n$  si  $n$  dénote le nombre d'horloges.  $\mathcal{R}$  est donc fini et de taille exponentielle dans la taille de l'automate temporisé. De plus, la relation d'équivalence satisfait les propriétés suivantes :

$$v \approx v' \implies \begin{cases} \text{pour toute contrainte } g \text{ de } \mathcal{A}, v \models g \iff v' \models g \\ \forall d \in \mathbb{T}, \exists d' \in \mathbb{T} \text{ tel que } v + d \approx v' + d' \end{cases}$$

**Exemple 1.18** (Régions en dimension 2). Considérons un ensemble de deux horloges  $x$  et  $y$  et fixons comme constante maximale la valeur 3. L'ensemble de régions correspondant est alors représenté sur la figure ci-contre.



**Définition 1.19** (Automate des régions). *Nous pouvons alors définir l'automate des régions de  $\mathcal{A}$  comme le système de transitions fini  $\mathcal{R}(\mathcal{A}) = (\Gamma, \gamma_0, \rightarrow)$  où*

- $\Gamma = L \times \mathcal{R}$  est l'ensemble  $\{(\ell, r) \mid \ell \in L, r \in \mathcal{R}\}$ ,



- $\gamma_0$  est l'état initial  $(\ell_0, r_0)$  où  $r_0$  est la région qui contient la valuation  $\mathbf{0}$ ,
- $\rightarrow \subseteq \Gamma \times (\Sigma \cup \{\tau\}) \times \Gamma$  et  $((\ell, r), \sigma, (\ell', r')) \in \rightarrow$  ssi  $(\ell, r) \neq (\ell', r')$  et
  - ou bien  $\sigma \in \Sigma$  et  $(\ell, v) \xrightarrow{\sigma} (\ell', v')$  est une transition de  $\llbracket \mathcal{A} \rrbracket$  pour un certain  $v \in r$  et  $v' \in r'$ ,
  - ou bien  $\sigma$  est le symbole  $\tau$ , et il existe  $d \in \mathbb{T}$  tel que  $(\ell, v) \xrightarrow{d} (\ell, v')$  est une transition de  $\llbracket \mathcal{A} \rrbracket$  pour un certain  $v \in r$  et un certain  $v' \in r'$ .

De plus, l'ensemble des états finals (respectivement répétés – nous considérons une condition de Büchi simple) de  $\mathcal{R}(\mathcal{A})$ , noté  $\Gamma_f$  (respectivement  $\Gamma_r$ ), est défini par :

$$\Gamma_f = \{(\ell, r) \mid \ell \in F, r \in \mathcal{R}\}$$

$$\Gamma_r = \{(\ell, r) \mid \ell \in R, r \in \mathcal{R}\}$$

La propriété fondamentale de l'automate des régions est qu'il est en *bisimulation à temps abstrait* avec l'automate temporisé  $\mathcal{A}$ . Cette notion correspond à la notion de bisimulation introduite précédemment pour les systèmes de transitions temporisés dans laquelle pour les transitions de délais, seule la nature de la transition est conservée par la simulation, et pas la valeur exacte du délai. Plus précisément, cette propriété s'énonce ainsi : nous définissons la relation  $\simeq \subseteq (L \times \mathbb{T}^X) \times \Gamma$  par :

$$(\ell, v) \simeq (\ell, r) \iff [v] = r.$$

Fixons  $(\ell, v) \in L \times \mathbb{T}^X$ , et  $(\ell, r) \in \Gamma$  telles que  $(\ell, v) \simeq (\ell, r)$ . Alors nous avons :

- $\forall a \in \Sigma, (\ell, v) \xrightarrow{a} (\ell', v') \iff (\ell, r) \xrightarrow{a} (\ell', [v'])$ ,
- $\forall t \in \mathbb{T}, (\ell, v) \xrightarrow{t} (\ell, v+t) \iff (\ell, r) \xrightarrow{\tau} (\ell, [v+t])$ .

Cette relation préserve les transitions discrètes. En revanche, pour les transitions de délai, elle préserve leur nature, mais pas la valeur du délai.

Il est alors possible de démontrer à l'aide de cette propriété, et grâce à la borne sur le nombre de régions, que le problème de l'accessibilité est dans la classe **PSPACE** pour les automates temporisés. Alur et Dill ont même démontré que cette complexité est optimale :

**Théorème 1.20** ([AD90, AD94]). *Tester le vide du langage accepté par un automate temporisé d'Alur et Dill est un problème PSPACE-complet.*

## 1.5 Extensions du modèle d'Alur et Dill

Nous présentons dans cette section quelques extensions classiques des automates temporisés, ainsi que les résultats fondamentaux les concernant. Nous étudierons dans les parties suivantes des problèmes faisant intervenir ces différentes extensions du modèle initial.

### 1.5.1 Transitions silencieuses

Il est possible d'étendre la définition des automates temporisés en autorisant l'étiquette  $\varepsilon$  pour les transitions. Cette extension naturelle avait déjà été envisagée dans [AD90]. Nous noterons dans la suite  $AT_\varepsilon$  la classe des automates temporisés avec transitions silencieuses. Comme cela a été fait pour les systèmes de transitions temporisés, il est facile d'étendre la sémantique des automates temporisés au cas des transitions silencieuses, en utilisant la projection  $\pi_\Sigma$ . De même, il est aisé de vérifier que le problème du vide est encore décidable pour la classe  $AT_\varepsilon$  puisque l'étiquette des transitions n'a pas d'influence sur un problème d'accessibilité. Enfin, nous définissons naturellement la classe des

langages temporisés  $\varepsilon$ -réguliers comme l'ensemble des langages temporisés reconnus par un automate temporisé avec transitions silencieuses. [BDGP98] présente une synthèse de différents travaux visant à étudier l'expressivité de la classe  $AT_\varepsilon$ . Nous reprenons ici deux résultats qui nous seront utiles.

**Théorème 1.21** (Proposition 2 de [BDGP98]). *La classe  $AT$  est strictement moins expressive que la classe  $AT_\varepsilon$  du point de vue des langages, i.e.  $AT \subsetneq AT_\varepsilon$ .*

L'automate utilisé dans la preuve de la proposition 1.21 est étudié dans l'exemple 1.22.

**Exemple 1.22** (Un automate temporisé avec transitions silencieuses). L'automate temporisé avec transitions silencieuses représenté sur la figure 1.2 reconnaît le langage de mots temporisés finis suivant :

$$\mathcal{R}_{pair} = \{(a, \tau_1)(a, \tau_2) \dots (a, \tau_n) \mid \tau_i \equiv 0 \pmod{2} \text{ pour tout } 1 \leq i \leq n\}.$$

Dans [BDGP98], il est démontré que ce langage n'est reconnu par aucun automate temporisé sans transitions silencieuses.  $\lrcorner$

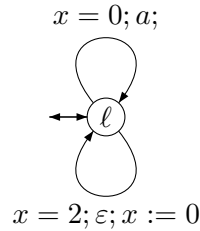


FIG. 1.2 – Un automate temporisé avec transitions silencieuses équivalent à aucun automate temporisé

Contrairement au cadre des automates finis, les transitions silencieuses permettent donc d'accroître strictement le pouvoir d'expression du modèle. Le résultat suivant précise une condition suffisante pour l'élimination des transitions silencieuses.

**Théorème 1.23** (Théorème 15 de [BDGP98]). *Soit  $\mathcal{A}$  un automate temporisé avec transitions silencieuses vérifiant les hypothèses suivantes :*

- aucun cycle ne contient d' $\varepsilon$ -transition avec remise à zéro,
- le langage accepté est non Zeno, i.e.  $\mathcal{L}^\omega(\mathcal{A}) \subseteq \mathcal{MT}^{\omega_{nz}}(\Sigma)$ .

*Alors il existe un automate temporisé sans transitions silencieuses  $\mathcal{A}'$  vérifiant  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ .*

Cependant, ce résultat laisse ouverte la question de la décidabilité du problème suivant : étant donné un automate temporisé avec transitions silencieuses, existe-t-il un automate temporisé sans transitions silencieuses reconnaissant le même langage ? Dans le chapitre 3, nous étudions ce problème ainsi que d'autres problèmes relatifs à l'introduction des transitions silencieuses dans les automates temporisés.

### 1.5.2 Mises à jour d'horloges

Une seconde extension concerne les mises à jour attachées aux transitions. Dans le modèle original, seules des opérations de remises à zéro sont autorisées. De nombreux autres types de mises à jour ont été envisagés par la suite, principalement dans [BDFP04]. Nous présentons ici deux de ces extensions que nous serons amenés à étudier par la suite, ainsi que les résultats correspondants de décidabilité et d'expressivité.

**Mises à jour intégrales.** Cette extension consiste à autoriser lors des transitions des opérations de la forme  $x := c$ , où  $c$  dénote un entier naturel. Formellement, pour un ensemble d'horloges  $X$ , nous considérons des applications  $\mu \in (\mathbb{N} \cup \{\perp\})^X$ . Étant donnée une valuation d'horloges  $v \in \mathbb{T}^X$ , la nouvelle valuation d'horloges  $v'$  obtenue après la mise à jour de  $v$  par  $\mu$  est définie ainsi :

$$\forall x \in X, v'(x) = \begin{cases} \mu(x) & \text{si } \mu(x) \neq \perp \\ v(x) & \text{sinon.} \end{cases}$$

Dans la suite, nous dénoterons par  $\text{AT}_{mi}$  la classe des automates temporisés d'Alur et Dill étendus avec les mises à jour intégrales.

**Mises à jour non déterministes.** Il est également possible d'autoriser les mises à jour d'horloges non déterministes. Les opérations considérées sont cette fois de la forme  $x : \in I$ , où  $I$  dénote un intervalle de temps. Formellement, pour un ensemble d'horloges  $X$ , nous considérons des applications  $\mu \in (\mathcal{I} \cup \{\perp\})^X$ . Étant donnée une valuation d'horloges  $v \in \mathbb{T}^X$ , la nouvelle valuation d'horloges  $v'$  obtenue après la mise à jour de  $v$  par  $\mu$  doit vérifier :

$$\forall x \in X, \begin{cases} v'(x) \in \mu(x) & \text{si } \mu(x) \neq \perp \\ v'(x) = v(x) & \text{sinon.} \end{cases}$$

Intuitivement, si l'opération de mise à jour est  $x : \in [0, 1[$ , la nouvelle valeur de l'horloge  $x$  est choisie de façon non déterministe dans l'intervalle  $[0, 1[$ . La classe des automates temporisés d'Alur et Dill étendus avec les mises à jour non déterministes sera notée  $\text{AT}_{mnd}$ . Il est facile de constater que la classe  $\text{AT}_{mnd}$  contient la classe précédente  $\text{AT}_{mi}$ . Nous présentons certains résultats issus de [BDFP04] précisant l'expressivité de ces différentes classes, ainsi que leur décidabilité.

**Théorème 1.24** (Expressivité, [BDFP04]). *Les pouvoirs expressifs des deux classes  $\text{AT}_{mi}$  et  $\text{AT}_{mnd}$  vérifient les propriétés suivantes :*

- les classes  $\text{AT}_{mi}$  et  $\text{AT}$  sont (fortement) bisimilaires,
- $\text{AT}_{mnd} \subseteq_{*, \omega_n, Z} \text{AT}_\varepsilon$

**Théorème 1.25** (Décidabilité, [BDFP04]). *Les classes d'automates temporisés  $\text{AT}_{mi}$  et  $\text{AT}_{mnd}$  sont décidables. Tester le problème du vide pour ces classes est PSPACE-complet.*

L'extension  $\text{AT}_{mi}$ , « gratuite » du point de vue de la vérification, permet donc plus de souplesse dans la modélisation, sans ajouter à l'expressivité du modèle. De plus, elle permet d'apporter de la concision au modèle, comme cela a été démontré récemment.

**Théorème 1.26** (Concision, Théorème 6 de [BC07]). *La classe  $\text{AT}_{mi}$  est exponentiellement plus concise que la classe  $\text{AT}$  des automates temporisés d'Alur et Dill.*

Nous nous intéressons à la classe  $\text{AT}_{mi}$  dans le chapitre 4 afin de proposer une nouvelle approche pour l'analyse de ces modèles fondée sur une traduction vers une extension temporisée des réseaux de Petri. Dans le chapitre 5, nous obtenons de nouveaux résultats d'expressivité pour la classe  $\text{AT}_{mnd}$  qui complètent le deuxième point du théorème 1.24. Ces résultats sont obtenus à l'aide d'une traduction de ces modèles dans une nouvelle classe de réseaux de Petri temporisés.

### 1.5.3 Contraintes d'horloges diagonales

Une autre extension porte sur la nature des contraintes qui peuvent être utilisées pour étiqueter les transitions. L'extension, déjà envisagée dans [AD90], consiste à autoriser l'ensemble le plus général  $\mathcal{C}(X)$  de contraintes portant sur l'ensemble d'horloges  $X$ . Rappelons que la seule différence concerne la présence de contraintes diagonales de la forme  $x - y \sim c$ . La classe des automates temporisés d'Alur et Dill étendus avec les contraintes diagonales sera notée  $AT_{gd}$ .

Nous synthétisons ici les résultats relatifs à cette extension, et les présentons dans deux catégories différentes, l'expressivité d'une part, et la décidabilité d'autre part.

**Expressivité des contraintes diagonales.** Les auteurs de [AD90] mentionnaient dans leurs travaux que le modèle des automates temporisés avec gardes diagonales n'est pas plus expressif que le modèle standard. Une preuve complète de ce résultat a été écrite dans [BDGP98]. Nous donnons ici le schéma de cette preuve, car il est utile pour la présentation de nos travaux.

**Théorème 1.27** (Expressivité, Lemme 5 de [BDGP98]). *Soit  $\mathcal{A} \in AT_{gd}$  un automate temporisé avec gardes diagonales. Alors il existe un automate temporisé  $\mathcal{B}$ , sans gardes diagonales, tel que  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ .*

**Preuve.** (Schéma)

La transformation est présentée pour une garde diagonale. Elle doit ensuite être appliquée à chacune des gardes diagonales qui apparaît dans  $\mathcal{A}$  afin d'obtenir  $\mathcal{B}$ . Nous considérons donc le retrait d'une garde diagonale  $x - y \leq c$  de  $\mathcal{A}$ , avec  $c \geq 0$ . L'idée de la construction consiste à utiliser deux copies  $\mathcal{A}_\top$  et  $\mathcal{A}_\perp$  de l'automate, d'ensembles d'états associés  $L_\top$  et  $L_\perp$ , chacune des deux copies codant la valeur de vérité du booléen  $x - y \leq c$ . Il est facile de constater que ce booléen est invariant par écoulement du temps, et que la seule opération qui peut modifier sa valeur est une remise à zéro de  $x$  ou  $y$ . Ainsi, on constate qu'après la remise à zéro de  $x$ , on a :

$$x - y \leq c \iff -y \leq c \iff y > -c \iff \text{tt}$$

De façon similaire, après la remise à zéro de  $y$ , on a :

$$x - y \leq c \iff x \leq c$$

Ceci induit une construction naturelle, qui va passer d'une copie à l'autre de l'automate en fonction du test indiqué dans les équivalences précédentes. Cette construction est représentée sur la figure 1.3.

La preuve de correction de cette construction repose sur la propriété énoncée de façon informelle plus haut portant sur les configurations accessibles  $(\ell, v)$ , prouvée par récurrence sur la longueur de l'exécution de l'automate :

$$\ell \in L_\top \implies v \models x - y \leq c$$

$$\ell \in L_\perp \implies v \models x - y > c$$

□

Une étude simple de la construction proposée montre, en notant respectivement  $L_{\mathcal{A}}$  et  $L_{\mathcal{B}}$  les états de contrôle de  $\mathcal{A}$  et  $\mathcal{B}$ , que les deux automates  $\mathcal{A}$  et  $\mathcal{B}$  vérifient :

$$|L_{\mathcal{B}}| = 2^n \times |L_{\mathcal{A}}|$$

où  $n$  est le nombre de gardes diagonales distinctes de  $\mathcal{A}$ . Ainsi, cette construction induit une explosion de la taille du modèle, et laisse penser que les gardes diagonales apportent de la concision au modèle. Ceci a été prouvé récemment, et est établi par le résultat suivant.

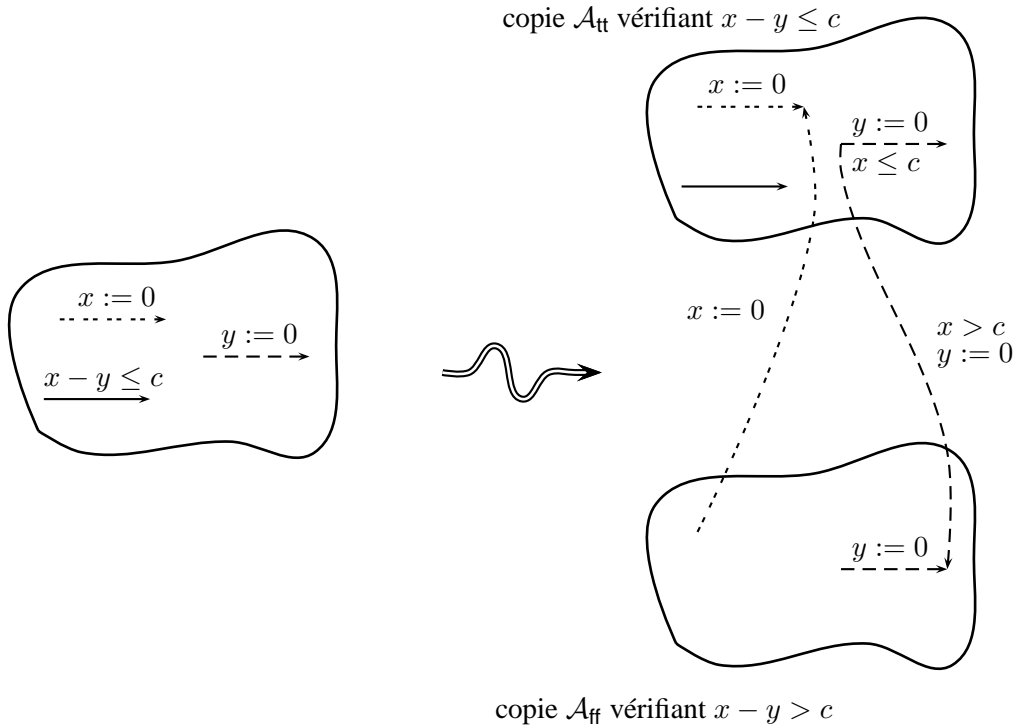


FIG. 1.3 – La construction permettant de retirer une contrainte diagonale.

**Théorème 1.28** (Concision, Théorème 2 de [BC07]). *La classe  $AT_{gd}$  des automates temporisés étendus avec les contraintes diagonales est exponentiellement plus concise que la classe des automates temporisés d'Alur et Dill.*

**Décidabilité des automates temporisés avec gardes diagonales.** Les résultats d'expressivité précédents impliquent en particulier que la classe des automates temporisés étendus avec les gardes diagonales est décidable. Cependant, il est également possible de modifier légèrement la définition des régions de sorte à définir un nouvel ensemble de régions, compatibles avec les gardes diagonales (*i.e.* que les propriétés énoncées en section 1.4 sur les régions sont vérifiées). Considérons la relation d'équivalence  $\approx$  définie sur l'ensemble des valuations par  $v \approx v'$  si et seulement si chacune des conditions suivantes est satisfaite :

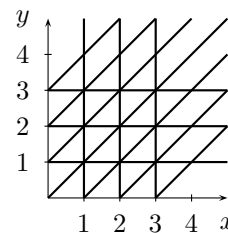
- pour toute horloge  $x \in X$ , ou bien  $v(x)$  et  $v'(x)$  sont strictement supérieures à  $K$ , ou bien  $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ ,
- pour toutes horloges  $x, x' \in X$ , si la valeur absolue  $|v(x) - v(x')|$  est inférieure ou égale à  $K$ , alors
  - $\langle v(x) \rangle \leq \langle v(x') \rangle$  ssi  $\langle v'(x) \rangle \leq \langle v'(x') \rangle$  et
  - $\langle v(x) \rangle = 0$  ssi  $\langle v'(x) \rangle = 0$ .

À nouveau, cette relation est bien une relation d'équivalence et ses classes d'équivalences sont appelées des régions. Nous notons cette fois  $\mathcal{R}_d$  l'ensemble de ces régions. La seule différence par

rapport à l'ensemble  $\mathcal{R}$  réside dans le deuxième point des conditions précédentes. Celui-ci ne concernait que les valuations pour lesquelles nous avons à la fois  $v(x) \leq K$  et  $v(x') \leq K$ . Dans cette nouvelle définition, les contraintes portent sur toutes les valuations vérifiant  $|v(x) - v(x')| \leq K$ . Graphiquement, ceci apparaît sur l'exemple 1.30 ci-dessous par le prolongement des lignes diagonales au-delà du carré  $[0, K] \times [0, K]$ . Ceci permet de tenir compte de la présence des gardes diagonales et nous pouvons ainsi étendre les résultats précédents pour l'ensemble  $\mathcal{R}$  à l'ensemble  $\mathcal{R}_d$  vis-à-vis des automates temporisés avec gardes diagonales. Nous obtenons ainsi le théorème suivant :

**Théorème 1.29** (Décidabilité, [AD90, AD94]). *La classe  $AT_{gd}$  des automates temporisés étendue avec les gardes diagonales est décidable. Tester le problème du vide pour cette classe est PSPACE-complet.*

**Exemple 1.30** (Régions de  $\mathcal{R}_d$  en dimension 2). Comme plus haut, nous considérons un ensemble de deux horloges  $x$  et  $y$  et fixons comme constante maximale la valeur 3. La partition en régions correspondant à l'ensemble  $\mathcal{R}_d$  est alors représentée sur la figure ci-contre.



Les contraintes diagonales permettent donc d'apporter de la concision au modèle, sans augmenter la complexité du problème du vide. De plus, elles peuvent être utiles lors de la phase de modélisation. Ainsi, les contraintes diagonales ont été utilisées dans [FPY02] pour des applications d'ordonnancement. Or la transformation qui permet de s'affranchir des gardes diagonales est coûteuse, et inévitable du fait des résultats de concision. Il est donc pertinent de s'intéresser à cette classe afin d'obtenir des algorithmes dédiés pour vérifier ce type de modèles. Nous présenterons deux solutions, l'une issue des réseaux de Petri temporels dans le chapitre 4, l'autre développée spécifiquement pour les automates temporisés avec gardes diagonales dans le chapitre 6.

## 1.6 Réseaux d'automates temporisés

Plusieurs modèles peuvent être envisagés pour la composition d'automates temporisés. Nous présentons les réseaux d'automates temporisés, définis comme une application de la synchronisation de systèmes de transitions introduite précédemment. Ce modèle est fondé sur la synchronisation des différents automates sur les étiquettes des actions et sur la synchronisation des échelles de temps.

**Définition 1.31** (Réseau d'automates temporisés). *Un réseau d'automates temporisés est constitué de la donnée d'une famille finie de  $n$  automates temporisés  $(\mathcal{A}_i)_{1 \leq i \leq n}$  sur  $\Sigma$  et d'une fonction  $f$  de synchronisation  $n$ -aire définie sur  $\Sigma$ . La sémantique d'un tel réseau est la synchronisation des systèmes de transitions temporisés  $\llbracket \mathcal{A}_i \rrbracket$  de chacun des automates temporisés vis-à-vis de la fonction de synchronisation  $f$ . Étant donné un réseau d'automates temporisés  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$ , nous noterons  $\llbracket \mathcal{A} \rrbracket$  le système de transitions correspondant.*

Comme pour la synchronisation de systèmes de transitions temporisés, la fonction de synchronisation décrit donc les interactions discrètes entre les processus tandis que les transitions de délais synchronisent tous les automates puisque tous doivent évoluer à la même vitesse.

**Exemple 1.32** (Un réseau d'automates temporisés modélisant un passage à niveau.). Nous représentons ici l'exemple classique du passage à niveau [AD94] modélisé à l'aide d'un réseau d'automates

temporisés. Cet exemple fait intervenir trois automates. Un premier automate modélise le comportement d'un train vis-à-vis du passage à niveau, un second modélise le comportement de la barrière, et un troisième représente le comportement d'un contrôleur qui commande les déplacements de la barrière en fonction de l'arrivée des trains. Ces automates sont représentés sur la figure 1.4. Dans les compositions envisagées, le nombre de trains est variable. Dans cet exemple, seules des synchronisations binaires sont utilisées. La synchronisation que nous avons définie plus haut est la plus générale et permet de coder toutes les compositions habituellement utilisées comme par exemple la synchronisation binaire. Dans le cas d'une composition impliquant deux trains, nous noterons le réseau d'automates de la façon suivante :

$$\mathcal{A} = (\text{TRAIN}_1 \parallel \text{TRAIN}_2 \parallel \text{BARRIÈRE} \parallel \text{CONTRÔLEUR})$$

pour indiquer la composition parallèle des différents automates. La fonction de synchronisation à considérer est la suivante :

$$f : \begin{cases} \text{TRAIN}_1, & \text{TRAIN}_2, & \text{BARRIÈRE}, & \text{CONTRÔLEUR}, \\ \text{App!} & \perp & \perp & \text{App?} & \mapsto & \text{App} \\ \text{Pars!} & \perp & \perp & \text{Pars?} & \mapsto & \text{Pars} \\ \perp & \text{App!} & \perp & \text{App?} & \mapsto & \text{App} \\ \perp & \text{Pars!} & \perp & \text{Pars?} & \mapsto & \text{Pars} \\ \perp & \perp & \text{Haut?} & \text{Haut!} & \mapsto & \text{Haut} \\ \perp & \perp & \text{Bas?} & \text{Bas!} & \mapsto & \text{Bas} \end{cases}$$

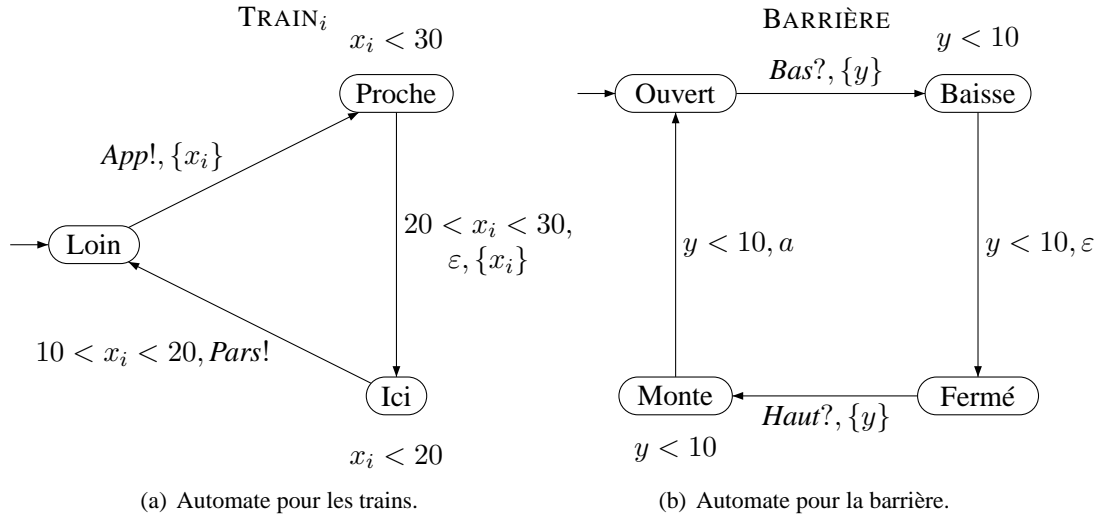
Remarquons enfin que dans cet exemple, chaque automate a sa propre horloge (aucune horloge n'est partagée.) De plus, les gardes égales à tt et les mises à jour égales à  $\emptyset$  n'ont pas été représentées afin d'alléger la figure.  $\lrcorner$

**Horloges partagées.** Dans la définition précédente, la composition des automates temporisés est réalisée uniquement à travers les actions réalisées. Ainsi, l'état de chaque automate est tout à fait local. Il est également intéressant de chercher à enrichir ce modèle en autorisant un certain partage de variables entre les automates temporisés. Nous considérons ici le cadre des horloges partagées. Ainsi, certaines horloges peuvent appartenir à l'ensemble des horloges de différents automates temporisés. En particulier, ceci implique qu'un automate temporisé peut mettre à jour des variables utilisées par d'autres automates. Afin de présenter ce cadre, nous ne pouvons pas nous conformer au cadre de la définition 1.5 de la synchronisation de systèmes de transitions temporisés.

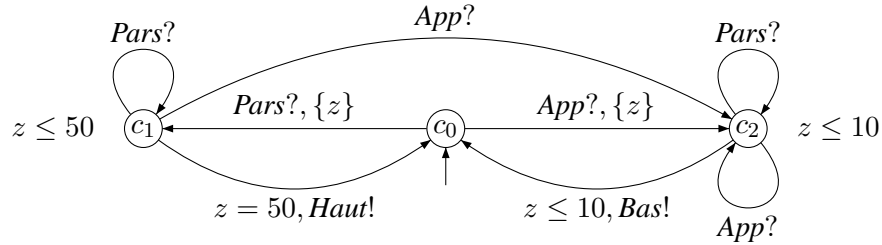
**Définition 1.33** (Réseau d'automates temporisés à horloges partagées). *Un réseau d'automates temporisés à horloges partagées est constitué de la donnée d'un ensemble d'horloges partagées  $X_0$ , d'une famille finie de  $n$  automates temporisés<sup>3</sup>  $(\mathcal{A}_i)_{1 \leq i \leq n}$  sur  $\Sigma$  dont les ensembles d'horloges respectifs  $X_i$  vérifient  $X_0 \subseteq X_i$  et d'une fonction  $f$  de synchronisation  $n$ -aire définie sur  $\Sigma$ . La sémantique d'un tel réseau  $\mathcal{A}$  est le système de transitions temporisé  $\llbracket \mathcal{A} \rrbracket = (Q, q_0, \rightarrow)$  défini par :*

- $Q = \{((\ell_1, v_1), \dots, (\ell_n, v_n)) \in \prod_{1 \leq i \leq n} (L_i \times \mathbb{T}^{X_i}) \mid \forall x \in X_0, \forall 1 \leq i, j \leq n, v_i(x) = v_j(x)\}$ ;
- $q_0 = ((\ell_{1,0}, \mathbf{0}), \dots, (\ell_{n,0}, \mathbf{0}))$ ;
- $\rightarrow$  est définie, pour tous  $q = ((\ell_i, v_i)_{1 \leq i \leq n})$ ,  $q' = ((\ell'_i, v'_i)_{1 \leq i \leq n}) \in Q$ , par :
  - **transitions discrètes** : pour tout  $a \in \Sigma$ ,  $q \xrightarrow{a} q'$  si et seulement si il existe un élément  $(a_i)_{1 \leq i \leq n} \in (\Sigma \cup \{\perp\})^n$  tel que :

<sup>3</sup>Nous prenons la notation naturelle  $\mathcal{A}_i = (\Sigma, X_i, L_i, T_i, \text{Inv}_i, \ell_{i,0}, L_{i,f}, L_{i,\tau})$ .



CONTRÔLEUR



(c) Automate pour le contrôleur.

FIG. 1.4 – Modélisation d'un passage à niveau.

- $f((a_i)_{1 \leq i \leq n}) = a$ ,
- pour tout indice  $i$  tel que  $a_i \neq \perp$ ,  $(\ell_i, v_i) \xrightarrow{a_i} (\ell'_i, v'_i)$ ,
- pour tout indice  $i$  tel que  $a_i = \perp$ ,  $\ell'_i = \ell_i$  et  $v'_i(x) = v_i(x)$  pour toute horloge  $x \in X_i \setminus X_0$ .
- **transitions de délai** : pour tout  $d \in \mathbb{T}$ ,  $q \xrightarrow{d} q'$  si et seulement si pour tout indice  $i$ ,  $(\ell_i, v_i) \xrightarrow{d} (\ell'_i, v'_i)$ .

Décrivons les différences existant avec le cadre précédent. D'abord, nous nous restreignons à un certain sous-ensemble des états du produit composé des états locaux dont les valuations s'accordent sur l'ensemble  $X_0$  des horloges partagées :

$$((\ell_1, v_1), \dots, (\ell_n, v_n)) \in Q \Rightarrow \forall x \in X_0, \forall 1 \leq i, j \leq n, v_i(x) = v_j(x) \quad (1.1)$$

De plus, lors d'une transition discrète, nous n'imposons la conservation de l'état pour les automates correspondant au symbole  $\perp$  que sur leur partie « privée », *i.e.* leur état de contrôle et leurs horloges privées ( $X_i \setminus X_0$ ). Leur valuation sur les horloges de  $X_0$  est parfaitement définie par les automates qui ont franchi une transition discrète et par la propriété (1.1). Notons tout de même que ceci requiert qu'au moins un automate a franchi une transition discrète, *i.e.* que cela interdit l'élément  $\perp^n$ , ce qui n'est pas restrictif. De plus, soulignons qu'il suffit qu'un des automates remette à zéro une horloge partagée pour que celle-ci soit remise à zéro. Naturellement, d'autres modèles sont envisageables



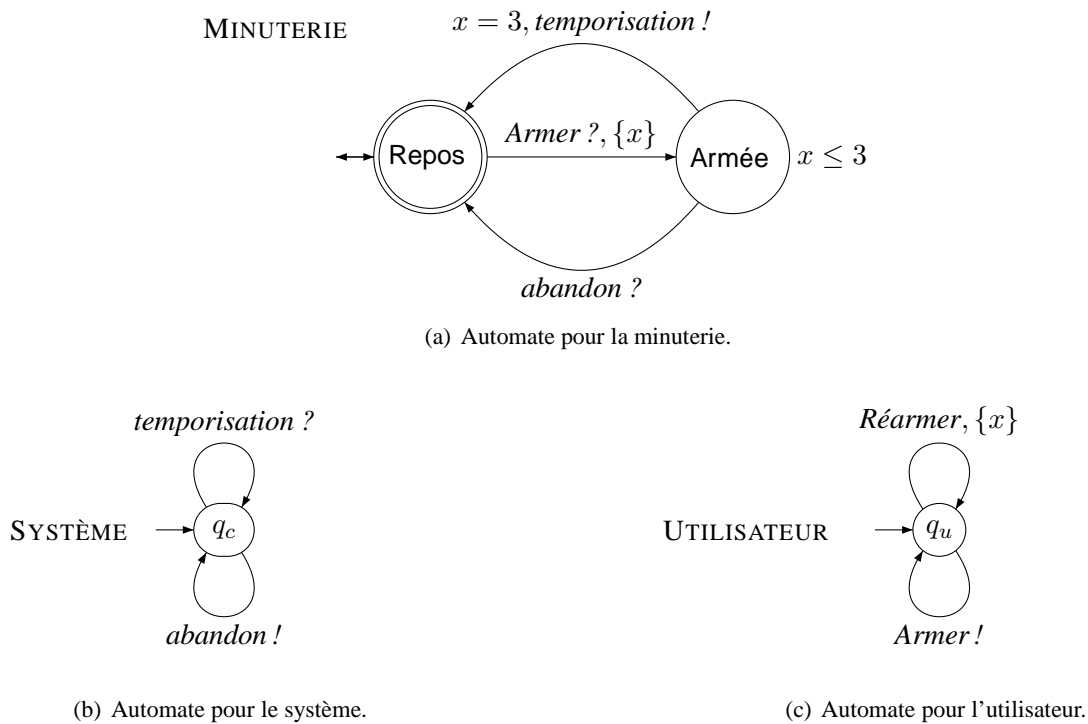


FIG. 1.5 – Modélisation d'un système contrôlé par une minuterie.

comme par exemple autoriser une horloge à n'être partagée que par deux processus, mais notre modèle est plus général.

**Exemple 1.34** (Réseau à horloges partagées). Nous présentons une extension naturelle de l'exemple présenté page 37 impliquant une horloge partagée. Ce réseau est représenté sur la figure 1.5. L'intuition de cet exemple est qu'un utilisateur peut à tout moment réarmer la minuterie ce qui se traduit par la remise à zéro de l'horloge  $x$ . D'autre part, le système contrôlé par cette minuterie interagit avec elle à travers les actions *temporisation ?* et *abandon !*. La composition considérée dans cet exemple est la suivante :

$$\mathcal{A} = (\text{MINUTERIE} \parallel \text{SYSTÈME} \parallel \text{UTILISATEUR})$$

┘

**Réseaux d'automates temporisés contre automates temporisés.** Nous avons présenté deux définitions pour les réseaux d'automates temporisés, avec ou sans horloges partagées. Une question naturelle consiste à se demander si les modèles définissables avec ces nouveaux formalismes le sont aussi dans le formalisme des automates temporisés. En d'autres termes, il s'agit de comparer l'expressivité des automates temporisés avec celle des réseaux d'automates temporisés. La réponse est positive : ces modèles sont expressivement équivalents du point de vue des langages et même du point de vue de la bisimulation. En réalité, nous allons montrer que leurs systèmes de transitions temporisés sont isomorphes. Ce résultat n'enlève cependant pas l'intérêt de la description d'un système comme un réseau d'automates plutôt que comme un automate, car l'automate temporisé équivalent est de taille exponentielle. Une description sous-forme de réseau est donc plus concise et plus facile à réaliser si le système analysé est conçu avec un point de vue modulaire.

Nous présentons la définition de l'automate temporisé équivalent à un réseau d'automates temporisés avec horloges partagées, le cas des réseaux sans horloges partagées en découlant facilement.

**Proposition 1.35.** *Étant donné un réseau d'automates temporisés avec horloges partagées  $\mathcal{A} = (X_0, (\mathcal{A}_i)_{1 \leq i \leq n}, f)$ . Définissons l'automate temporisé  $\mathcal{A}' = (\Sigma, X, L, T, \text{Inv}, \ell_0, L_f, L_r)$  par :*

- $X = \bigcup_{1 \leq i \leq n} X_i$ ;
- $L = \prod_{1 \leq i \leq n} L_i$ ;
- $T \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ . Soit  $\ell = (\ell_i)_{1 \leq i \leq n}$  et  $\ell' = (\ell'_i)_{1 \leq i \leq n}$  deux éléments de  $L$ ,  $g \in \mathcal{C}(X)$ ,  $a \in \Sigma$  et  $R \in 2^X$ .  $t = (\ell, g, a, R, \ell')$  appartient à  $T$  si et seulement s'il existe  $(t_1, \dots, t_n) \in \prod_{1 \leq i \leq n} (T_i \cup \{\perp\})$  tels qu'en notant  $I = \{i \mid t_i \neq \perp\}$ , nous avons :
  - $\forall i \in I, t_i = (\ell_i, g_i, a_i, R_i, \ell'_i)$ ,
  - en posant  $a_i = \perp$  pour tout  $i \notin I$ ,  $f(a_1, \dots, a_n) = a$ ,
  - $g = \bigwedge_{i \in I} g_i$ ,  $R = \bigcup_{i \in I} R_i$ ,
  - pour tout  $i \notin I$ ,  $\ell'_i = \ell_i$ .
- $\text{Inv}((\ell_i)_{1 \leq i \leq n}) = \bigwedge_{1 \leq i \leq n} \text{Inv}_i(\ell_i)$ ,
- $\ell_0 = (\ell_i, 0)_{1 \leq i \leq n}$ ,
- $L_f = \prod_{1 \leq i \leq n} L_{i,f}$  et
- $L_r = \prod_{1 \leq i \leq n} L_{i,r}$ .

Alors les deux systèmes de transitions temporisés  $\llbracket \mathcal{A} \rrbracket$  et  $\llbracket \mathcal{A}' \rrbracket$  sont isomorphes.

La preuve de cette proposition découle facilement des définitions. Il est important de remarquer que l'automate temporisé ainsi défini est de taille exponentielle dans la taille de la description du réseaux d'automates temporisés. La transformation n'est pas du tout intéressante en pratique et il est donc très important de développer des algorithmes dédiés aux réseaux d'automates temporisés qui puissent tirer profit de cette représentation distribuée du système global.

**Remarque 1.36.** La proposition précédente nous permet d'étendre aux réseaux d'automates temporisés (avec horloges partagées) les définitions introduites pour les automates temporisés. Par exemple, nous noterons  $\text{Acc}(\mathcal{A})$  l'ensemble des configurations accessibles dans  $\mathcal{A}$ . ┘

**Rôle des invariants dans les réseaux d'automates temporisés.** Nous avons expliqué dans la sous-section 1.4.2 comment les invariants pouvaient être retirés dans un automate temporisé tout en préservant les propriétés d'accessibilité. Nous allons expliquer à présent pourquoi une transformation similaire ne peut pas être appliquée aux réseaux d'automates temporisés. Commençons par remarquer que d'après la proposition précédente, il est possible de transformer le réseau d'automates temporisés (avec ou sans horloges partagées) en un automate temporisé et ensuite d'appliquer la transformation présentée dans la sous-section 1.4.2 à ce nouvel automate. Nous obtenons ainsi un automate temporisé sans invariants possédant le même ensemble d'états de contrôle accessibles. Cependant, cette transformation n'est pas satisfaisante puisqu'elle nécessite de construire l'automate « produit » équivalent ce que nous souhaitons éviter pour des raisons de complexité. De plus, cette transformation perd l'aspect distribué, et il n'est plus possible ensuite de le retrouver.

Considérons un réseau d'automates temporisés  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$ . Nous souhaiterions donc pouvoir modifier chacun des automates temporisés localement (uniquement en retirant les invariant et en modifiant les gardes) afin d'obtenir un nouveau réseau sans invariants équivalent du point de vue de l'accessibilité des états de contrôle et du point de vue du tir des transitions discrètes. Nous donnons sur la figure 1.6 un exemple simple illustrant qu'une telle transformation ne peut pas exister. En effet, il est facile de vérifier que dans ce réseau, si la transition  $c_1$  est franchie, alors la transition  $b$  va avoir lieu et la transition  $a$  ne pourra pas intervenir. Si les invariants sont supprimés, il n'est pas possible

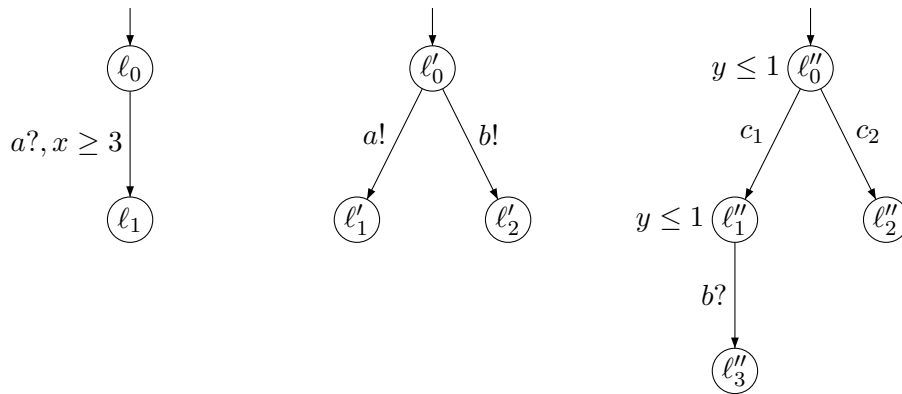


FIG. 1.6 – Un réseau d’automates temporisés nécessitant les invariants.

de renforcer les gardes des transitions et de conserver ce comportement. Ainsi, les occurrences des transitions discrètes ne sont donc pas conservées. De même, l’état de contrôle global  $(\ell_1, \ell'_1, \ell''_1)$  n’est pas accessible dans ce réseau mais le devient dans toute modification du réseau ne réduisant pas l’ensemble des états de contrôle accessibles. Ceci provient du fait que l’invariant sous lequel se trouve le système est une notion globale et qu’elle ne peut donc pas être traduite localement sans dupliquer certains états de contrôle.

Pour conclure, s’affranchir des invariants dans un réseau d’automates temporisés tout en préservant l’ensemble des états de contrôle accessibles ne peut pas être réalisé localement. Ceci peut paraître surprenant puisque les invariants sont en général mis en évidence par des relations de bisimulation et non par des équivalences de langage. S’affranchir des invariants nécessite donc la construction (coûteuse) de l’automate produit. Puisque cela n’est pas satisfaisant pour l’application de techniques de vérification tirant profit de l’aspect réparti d’un réseau d’automates temporisés, il est nécessaire de développer des techniques dédiées au modèle avec les invariants. Nous proposerons une solution à ce problème dans le chapitre 7.

## 1.7 Outils existants

Nous donnons ici un rapide aperçu de quatre outils qui comptent parmi les principaux outils existants pour la vérification de systèmes modélisés par des automates temporisés, ou des réseaux d’automates temporisés.

L’outil CMC (pour « Compositional Model Checking ») a été développé au Laboratoire Spécification et Vérification à l’ENS de Cachan par François Laroussinie depuis 1998. Le modèle analysé est celui des réseaux d’automates temporisés. Les propriétés vérifiées sont d’une nature assez riche et sont exprimées dans une logique nommée  $L_{\nu}$ . Cette logique est une logique modale temporisée autorisant l’opérateur de plus grand point fixe mais pas celui de plus petit point fixe. Les algorithmes de vérification mis en jeu reposent sur des techniques compositionnelles. Plus d’informations sur l’outil CMC sont accessibles à la page internet suivante :

<http://www.lsv.ens-cachan.fr/~fl/cmcweb.html>

L’outil HYTECH (pour « HYbrid Technology Tool ») a été développé à l’université de Berkeley, aux États-Unis, par Tom Henzinger, Pei-Hsin Ho et Howard Wong-Toi depuis 1997. Il n’est pas dédié

aux automates temporisés mais au modèle plus général des automates hybrides linéaires. Ceci rend certains des problèmes de vérification indécidables et les algorithmes implémentés dans cet outil sont en fait des semi-algorithmes. Cependant, son formalisme d'entrée plus général lui permet par exemple d'étudier des systèmes avec des paramètres, ce qui n'est pas possible avec les autres outils. Les propriétés vérifiées sont principalement des propriétés d'accessibilité et de sûreté. Les principaux travaux relatifs à cet outil peuvent être obtenus depuis la page internet de l'outil :

<http://embedded.eecs.berkeley.edu/research/hytech/>

L'outil KRONOS a été développé au laboratoire VERIMAG à Grenoble de 1997 à 2002. Les principaux auteurs sont Sergio Yovine, Alfredo Olivero, Conrado Daws et Stavros Tripakis. Il permet la vérification de propriétés temporisées de systèmes temporisés modélisés par des réseaux d'automates temporisés. Les propriétés vérifiées sont celles exprimées dans la logique TCTL obtenue comme une extension temporisée de la logique temporelle classique CTL. Les algorithmes implémentés dans cet outil sont l'analyse en avant, l'analyse en arrière et un algorithme pour TCTL. La page internet de l'outil est :

<http://www-verimag.imag.fr/TEMPORISE/kronos/>

L'outil UPPAAL (pour Uppsala et Aalborg) a été développé conjointement par les universités d'Uppsala, en Suède, et d'Aalborg, au Danemark depuis 1995. Les personnes ayant contribué à son développement sont trop nombreuses pour être citées ici, mais les deux principaux instigateurs de ce projet sont Wang Yi (Uppsala) et Kim Larsen (Aalborg). Ayant fait l'objet d'un effort de développement important, il intègre la plupart des dernières avancées théoriques existantes pour les automates temporisés, ainsi que de nombreuses heuristiques qui permettent d'améliorer sensiblement l'efficacité de l'outil. Cela en fait un des outils les plus utilisés. Le modèle d'entrée est une composition d'automates temporisés proche du modèle présenté ici. Des variables entières bornées peuvent également être utilisées. Les propriétés qui peuvent être vérifiées sont exprimables dans un fragment de la logique TCTL qui n'autorise pas les emboîtements d'opérateurs temporels. L'algorithme d'analyse en avant constitue le cœur de cet outil et la plupart des algorithmes sont des variantes de celui-ci. Enfin, plus récemment, diverses extensions ont été développées afin de proposer des outils pour d'autres objectifs que la vérification. La page internet de l'outil UPPAAL est :

<http://www.uppaal.com/>

## Chapitre 2

# Extensions temporisées des réseaux de Petri

Les réseaux de Petri [Pet62] constituent un modèle fondamental pour la modélisation de systèmes discrets. Ils sont utilisés dans de nombreux domaines, y compris dans les milieux industriels, et sont adaptés à la modélisation de systèmes distribués, ou répartis. Les applications sont diverses et couvrent les domaines des systèmes de production, de transports et de communication. De nombreuses extensions du modèle ont été proposées, et en particulier des extensions adaptées à la modélisation de systèmes temporisés. Dans ce chapitre, nous présenterons dans un premier temps en section 2.1 le modèle des réseaux de Petri, accompagné des définitions nécessaires. Ensuite, nous introduirons deux extensions temporisées de ce modèle, les réseaux de Petri temporels (en section 2.2), et les réseaux de Petri temporisés (en section 2.3). Ces deux extensions comptent parmi les plus couramment utilisées et sont au centre de certains de nos travaux.

### 2.1 Réseaux de Petri

Nous rappelons dans cette section quelques définitions classiques liées aux réseaux de Petri qui seront utiles ultérieurement. Nous introduisons ensuite le modèle des réseaux de Petri, et énonçons certaines de ses propriétés fondamentales.

#### 2.1.1 Préliminaires

Nous introduisons la notion de multi-ensemble et les différentes opérations qui leur sont associées. Étant donné un ensemble  $\mathcal{E}$ ,  $\text{MEns}(\mathcal{E})$  dénote l'ensemble des applications  $x$  de  $\mathcal{E}$  dans  $\mathbb{N}$  à support fini, *i.e.* telles que l'ensemble  $\text{dom}(x) = \{t \in \mathcal{E} \mid x(t) \neq 0\}$  est fini. Ces applications seront appelées des *multi-ensembles de  $\mathcal{E}$* <sup>1</sup>. De plus, si  $\mathcal{E}$  est fini, un élément de  $\text{MEns}(\mathcal{E})$  sera appelé un *marquage sur  $\mathcal{E}$* . Nous notons enfin  $\text{taille}(x) = \sum_{t \in \text{dom}(x)} x(t)$ . Dans le contexte des marquages, nous représenterons parfois ces objets sous la forme suivante :

$$\text{MEns}(\mathcal{E}) \ni x = \sum_{t \in \text{dom}(x)} x(t) \cdot t$$

Si  $x(t)$  vaut 1, nous écrirons simplement  $t$  au lieu de  $1 \cdot t$ . Nous présentons maintenant différentes opérations agissant sur les multi-ensembles.

---

<sup>1</sup>Nous prenons donc la convention que tous les multi-ensembles que nous considérerons sont à support fini.

**Somme.** Soit  $x, y \in \text{MEns}(\mathcal{E})$ , alors  $x + y \in \text{MEns}(\mathcal{E})$  est défini ainsi :

$$\forall e \in \mathcal{E}, (x + y)(e) = x(e) + y(e).$$

**Ordre et différence.** D'autre part, nous définissons la relation  $\leq$  par  $y \leq x$  si et seulement si  $\forall e \in \mathcal{E}, y(e) \leq x(e)$ . Si  $y \leq x$ , alors  $x - y \in \text{MEns}(\mathcal{E})$  est défini par :  $\forall e \in \mathcal{E}, (x - y)(e) = x(e) - y(e)$ .

**Écoulement du temps.** Pour  $d \in \mathbb{T}$  et  $x \in \text{MEns}(\mathbb{T})$ ,  $x + d \in \text{MEns}(\mathbb{T})$  est défini par

$$\forall \tau \in \mathbb{T}, (x + d)(\tau) = \begin{cases} 0 & \text{si } \tau < d, \\ x(\tau - d) & \text{si } \tau \geq d. \end{cases}$$

**Produit par un ensemble fini.** Si un ensemble  $X$  est fini, alors il est facile de vérifier que les ensembles  $\text{MEns}(\mathcal{E})^X$  et  $\text{MEns}(X \times \mathcal{E})$  sont en bijection. Par la suite, nous utiliserons indifféremment l'une ou l'autre des deux notations pour faciliter la compréhension.

**Projection.** Nous définissons une opération de projection. Soit  $x \in \text{MEns}(\mathcal{E}_1 \times \dots \times \mathcal{E}_n)$ , et choisissons un ensemble d'indices  $I = \{i_1, \dots, i_k\}$  inclus dans  $\{1, \dots, n\}$ . Alors le multi-ensemble  $\pi_{i_1, \dots, i_k}(x) \in \text{MEns}(\mathcal{E}_{i_1} \times \dots \times \mathcal{E}_{i_k})$  est défini par :

pour tout  $(e_{i_1}, \dots, e_{i_k}) \in \mathcal{E}_{i_1} \times \dots \times \mathcal{E}_{i_k}$ ,

$$\pi_{i_1, \dots, i_k}(x)(e_{i_1}, \dots, e_{i_k}) = \sum_{e_{j_1}, \dots, e_{j_{n-k}} \in \mathcal{E}_{j_1} \times \dots \times \mathcal{E}_{j_{n-k}}} x(e_1, \dots, e_n),$$

où  $(j_1, \dots, j_{n-k})$  est l'unique uplet de  $n - k$  indices ordonnés par ordre croissant tel que l'ensemble  $\{j_1, \dots, j_{n-k}\}$  vérifie  $\{j_1, \dots, j_{n-k}\} \cap I = \emptyset$ .

### 2.1.2 Définition

Nous pouvons maintenant donner la définition d'un réseau de Petri.

**Définition 2.1** (Réseau de Petri). *Un réseau de Petri  $\mathcal{N}$  sur  $\Sigma_\varepsilon$  est un uplet  $(P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  où :*

- $P$  est un ensemble fini de places,
- $T$  est un ensemble fini de transitions avec  $P \cap T = \emptyset$ ,
- $\bullet(\cdot) \in \text{MEns}(P)^T$  est l'application d'incidence arrière,
- $(\cdot)^\bullet \in \text{MEns}(P)^T$  est l'application d'incidence avant,
- $M_0 \in \text{MEns}(P)$  est un marquage de  $P$ , appelé le marquage initial et
- $\Lambda : T \rightarrow \Sigma_\varepsilon$  est la fonction d'étiquetage.

Le comportement d'un réseau de Petri est très intuitif. Une configuration d'un réseau de Petri est définie par une répartition de jetons dans les différentes places. Plus formellement, ceci correspond à un élément de  $\text{MEns}(P)$ , i.e. à un marquage sur  $P$ . Initialement, des jetons sont déposés dans les places suivant la répartition indiquée par  $M_0$ . Ensuite, pour qu'une transition  $t$  puisse être tirée, il faut et il suffit que suffisamment de jetons soient présents, comme indiqué par  $\bullet t$ . Le tir de  $t$  provoque alors la consommation des jetons indiqués par  $\bullet t$ , puis la production de nouveaux jetons, comme indiqué par  $t^\bullet$ .

Afin de définir les comportements (finis ou infinis) du réseau qui sont acceptants, nous équipons le réseau de Petri de conditions d'acceptation.

**Définition 2.2** (Conditions d'acceptation, relation de satisfaction). *Étant donné un réseau de Petri  $\mathcal{N}$  dont l'ensemble des places est  $P$ , une condition d'acceptation pour  $\mathcal{N}$ , notée  $\mathbf{Acc}$ , est un ensemble fini  $\mathbf{Acc} = \{\mathbf{acc}_1, \dots, \mathbf{acc}_p\}$  de formules engendrées par la grammaire suivante :*

$$\mathbf{acc} ::= \sum_{p \in I} p \sim n \mid \mathbf{acc}_1 \wedge \mathbf{acc}_2 \mid \mathbf{tt}$$

où  $n \in \mathbb{N}$ ,  $I \subseteq P$ ,  $\sim \in \{\leq, \geq\}$ .

Nous définissons la relation de satisfaction pour les conditions d'acceptation. Celle-ci est définie sur les configurations  $M \in \mathbf{MEns}(P)$  du réseau  $\mathcal{N}$  de façon inductive par :

$$\begin{cases} M \text{ satisfait } \sum_{p \in I} p \sim n & \iff \sum_{p \in I} M(p) \sim n \\ M \text{ satisfait } \mathbf{acc}_1 \wedge \mathbf{acc}_2 & \iff M \text{ satisfait } \mathbf{acc}_1 \text{ et } M \text{ satisfait } \mathbf{acc}_2 \\ M \text{ satisfait } \mathbf{tt} & \iff \mathbf{tt} \end{cases}$$

La sémantique d'un réseau de Petri est alors définie par un système de transitions.

**Définition 2.3** (Sémantique d'un réseau de Petri). *La sémantique d'un réseau de Petri  $\mathcal{N}$  défini comme plus haut est donnée par un système de transitions  $\llbracket \mathcal{N} \rrbracket = (Q, q_0, \rightarrow)$  défini ainsi :*

- $Q = \mathbf{MEns}(P)$  est l'ensemble des marquages sur  $P$ ,
- $q_0 = M_0$  est l'état initial,
- $\rightarrow$  est définie par  $M \xrightarrow{a} M'$ , pour  $M, M' \in Q$ , et  $a \in \Sigma_\varepsilon$ , si, et seulement si, il existe un élément  $t \in T$  tel que :
  - $\Lambda(t) = a$ ,
  - $\bullet t \leq M$  et
  - $M' = (M - \bullet t) + t^\bullet$

Pour définir les conditions d'acceptation de  $\llbracket \mathcal{N} \rrbracket$ , nous considérons une condition d'acceptation  $\mathbf{Acc} = \{\mathbf{acc}_1, \dots, \mathbf{acc}_p\}$  de  $\mathcal{N}$ . Nous définissons alors l'ensemble des configurations finales de  $\llbracket \mathcal{N} \rrbracket$ , noté  $Q_f$ , par :

$$Q_f = \{M \in \mathbf{MEns}(P) \mid \exists 1 \leq i \leq p \text{ tel que } M \text{ satisfait } \mathbf{acc}_i\}$$

et la condition de Büchi généralisée associée à  $\llbracket \mathcal{N} \rrbracket$ , notée  $\{Q_r^1, \dots, Q_r^p\}$ , par :

$$Q_r^i = \{M \in \mathbf{MEns}(P) \mid M \text{ satisfait } \mathbf{acc}_i\}$$

Enfin, les définitions de comportements et de langage accepté de  $\llbracket \mathcal{N} \rrbracket$  sont étendues à  $\mathcal{N}$ .

**Quelques sous-classes des réseaux de Petri.** Un réseau de Petri  $\mathcal{N}$  sera dit

- $k$ -borné si, pour tout marquage  $M$  accessible dans  $\llbracket \mathcal{N} \rrbracket$  et pour toute place  $p$  du réseau, il y a au plus  $k$  jetons dans  $p$ , i.e.  $M(p) \leq k$ ,
- borné s'il existe un  $k \in \mathbb{N}$  pour lequel  $\mathcal{N}$  est  $k$ -borné, et
- sauf si  $\mathcal{N}$  est 1-borné.

Afin d'illustrer ces définitions, nous donnons deux exemples de réseaux de Petri.

**Exemple 2.4** (Réseau de Petri reconnaissant un langage non régulier.). Nous présentons un premier exemple de réseau de Petri  $\mathcal{N}_1$  sur la figure 2.1. Celui-ci présente la propriété intéressante de reconnaître un langage non régulier. Ceci représente une différence importante avec les automates temporisés qui sera utilisée ultérieurement dans de ce document. En effet, le langage accepté par ce réseau de

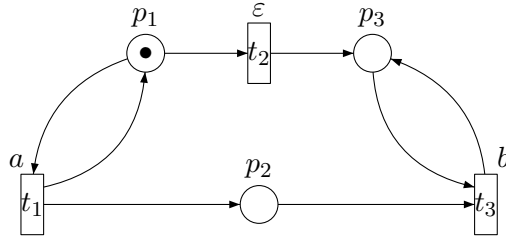


FIG. 2.1 – Réseau de Petri reconnaissant un langage non régulier.

Petri pour la condition d'acceptation  $\{\text{tt}\}$  est le langage  $\mathcal{L} = \{a^n b^m \mid 0 \leq m \leq n\}$  qui n'est pas un langage régulier.

La description formelle de  $\mathcal{N}_1 = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  est la suivante :

- $P = \{p_1, p_2, p_3\}$ ,
- $T = \{t_1, t_2, t_3\}$ ,
- $\Sigma = \{a, b\}$ ,
- $\bullet(\cdot)$  et  $(\cdot)^\bullet$  sont définies par :
  - $\bullet t_1 = \bullet t_2 = \{p_1\}$ ,
  - $t_1^\bullet = \{p_1, p_2\}$ ,
  - $t_2^\bullet = t_3^\bullet = \{p_3\}$ ,
  - $\bullet t_3 = \{p_2, p_3\}$ .
- $M_0 = p_1$  et
- $\Lambda$  est définie par :
  - $\Lambda(t_1) = a$ ,
  - $\Lambda(t_2) = \varepsilon$  et
  - $\Lambda(t_3) = b$ .

┘

**Exemple 2.5** (Réseau de Petri modélisant l'exclusion mutuelle). Un second exemple de réseau de Petri est représenté sur la figure 2.2. Cet exemple assure une propriété d'exclusion mutuelle entre les deux processus  $\mathcal{P}_1$  et  $\mathcal{P}_2$  qui peuvent accéder à une section critique (état SC). Par exemple, ceci peut correspondre à l'écriture d'une variable partagée.

Plus formellement, le réseau de Petri  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  représenté ici vérifie :

- $P = \{\text{SC}_1, \text{SC}_2, \text{Inactif}_1, \text{Inactif}_2, \text{Verrou}\}$ ,
- $T = \{t_1, t_2, t_3, t_4\}$ ,
- $\Sigma = \{\text{Entrée}, \text{Sortie}\}$ ,
- $\bullet(\cdot)$  et  $(\cdot)^\bullet$  sont définies par :
  - $\bullet t_1 = t_2^\bullet = \{\text{SC}_1\}$ ,
  - $\bullet t_2 = t_1^\bullet = \{\text{Inactif}_1, \text{Verrou}\}$ ,
  - $\bullet t_3 = t_4^\bullet = \{\text{Verrou}, \text{Inactif}_2\}$ ,
  - $\bullet t_4 = t_3^\bullet = \{\text{SC}_2\}$ .
- $M_0 = \text{Inactif}_1 + \text{Inactif}_2 + \text{Verrou}$  et
- $\Lambda$  est définie par :
  - $\Lambda(t_1) = \Lambda(t_4) = \text{Sortie}$ ,
  - $\Lambda(t_2) = \Lambda(t_3) = \text{Entrée}$ .

Dans cet exemple, il est facile de vérifier que le réseau est sain. De plus, le jeton de la place



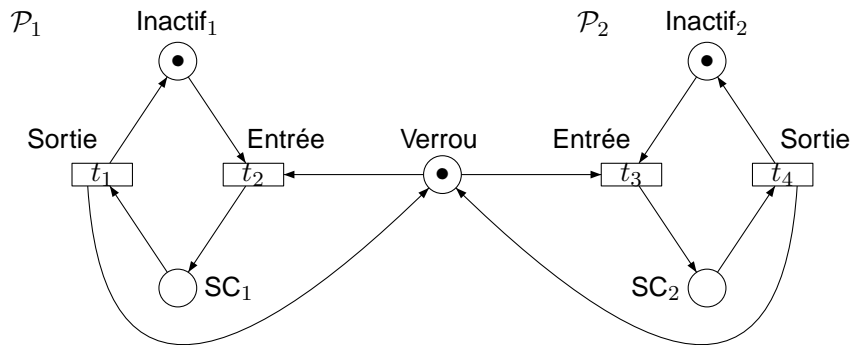


FIG. 2.2 – Réseau de Petri modélisant l'exclusion mutuelle.

Verrou permet l'accès à la section critique. Il est alors possible de démontrer une propriété d'exclusion mutuelle vérifiée par ce réseau : les processus  $\mathcal{P}_1$  et  $\mathcal{P}_2$  ne peuvent pas marquer simultanément la place SC. Plus formellement, pour tout marquage accessible  $M$ , nous avons :

$$M(\text{SC}_1) + M(\text{SC}_2) \leq 1.$$

┘

### 2.1.3 Résultats classiques

Nous rappelons à présent quelques résultats standards sur les réseaux de Petri afin d'illustrer la complexité des problèmes de vérification pour ce modèle.

Dans le contexte de la vérification, il est naturel de s'intéresser à la décidabilité de certaines propriétés. Un panorama de ce sujet a été proposé dans [EN94]. Nous mentionnons ici certains des résultats fondamentaux pour ce modèle. Notons que les bornes de complexité associées sont très élevées, ce qui implique que les problèmes similaires pour des extensions temporisées le seront également.

- L'accessibilité d'un marquage est décidable. Ce problème consiste à déterminer, étant donné un marquage  $M$ , s'il existe une exécution du réseau de Petri partant du marquage initial et atteignant le marquage cible  $M$ . Ce résultat a été démontré dans [Kos82, May84]. Ce problème est EXPSPACE-difficile [CLM76], mais les algorithmes proposés nécessitent un espace non primitif récursif.
- La couverture d'un marquage est décidable. Ce problème consiste à déterminer, étant donné un marquage  $M$ , s'il existe une exécution du réseau de Petri partant du marquage initial et atteignant un marquage  $M'$  plus grand que le marquage cible  $M$ , *i.e.* vérifiant  $M' \geq M$ . La décision de ce problème a été démontrée par Karp et Miller [KM69]. Leur algorithme met en jeu la construction d'un arbre dit arbre de couverture du réseau de Petri et nécessite un espace non primitif récursif. Cet algorithme a ensuite été amélioré par Rackoff [Rac78], démontrant ainsi que ce problème est EXPSPACE-complet, la borne inférieure ayant été obtenue dans [CLM76].
- La bornitude d'un réseau de Petri est décidable. Ce problème consiste à déterminer si le réseau est borné. La preuve de la décidabilité de ce problème repose également sur la construction de l'arbre de couverture. Ce problème est également EXPSPACE-complet.

Soulignons que dans le cadre des réseaux de Petri bornés et saufs, les complexités des problèmes sont réduites. Par exemple, l'accessibilité et la vivacité sont des problèmes PSPACE-complets pour les réseaux de Petri saufs.

## 2.2 Réseaux de Petri temporels

La première extension, appelée réseaux de Petri temporels et introduite dans [Mer74], consiste à ajouter une notion quantitative de temps dans le modèle par le biais des périodes d'activation des transitions. Il existe en fait plusieurs sémantiques pour cette extension qui dépendent de la définition prise pour cette période d'activation. Soulignons enfin que nous considérons la sémantique simple serveur, et non la sémantique multi-serveur (voir [BD01] pour une discussion sur ces deux sémantiques).

### 2.2.1 Définition

**Définition 2.6** (Réseaux de Petri temporels). *Un réseau de Petri temporel  $\mathcal{N}$  sur  $\Sigma_\varepsilon$  est un uplet  $(P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda, I)$  où :*

- $(P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  est un réseau de Petri,
- $I : T \mapsto \mathcal{I}$  associe à chaque transition un intervalle de tir <sup>2</sup>

**Sémantique.** Une *configuration* d'un réseau de Petri temporel est une paire  $(M, \nu)$ , où  $M$  est un *marquage* sur  $P$  pour le réseau de Petri sous-jacent, *i.e.* un élément de  $\text{MEns}(P)$ , avec  $M(p)$  le nombre de jetons présents dans la place  $p$ . Une transition  $t$  est *activée* dans le marquage  $M$  si  $M \geq \bullet t$ . Nous dénotons par  $\text{Actif}(M)$  l'ensemble des transitions actives dans  $M$ . La seconde composante de la paire  $(M, \nu)$  est une valuation sur  $\text{Actif}(M)$  qui associe à chaque transition active son *âge*, *i.e.* la quantité de temps qui s'est écoulée depuis que cette transition est active. Une transition active  $t$  est alors *franchissable* (ou *tirable*) si  $\nu(t)$  appartient à l'intervalle  $I(t)$ . Le résultat de ce tir est la nouvelle configuration  $(M', \nu')$ . Le nouveau marquage  $M'$  est le marquage habituel :  $M' = M - \bullet t + t^\bullet$ . D'autre part, l'âge de certaines transitions a été remis à zéro et nous dirons que les transitions correspondantes ont été *réactivées*. C'est à ce niveau que différentes sémantiques existent, selon le choix de définition de cette opération. Dans le reste de ces travaux, nous choisissons la sémantique standard [BD91, AL00]. Le lecteur intéressé par les autres sémantiques possibles peut trouver dans [BCH<sup>+</sup>05a] une comparaison approfondie des sémantiques alternatives. Le prédicat spécifiant quand  $t'$  est réactivée par le tir de  $t$  à partir du marquage  $M$  est défini ainsi :

$$\uparrow \text{Actif}(t', M, t) = t' \in \text{Actif}(M - \bullet t + t^\bullet) \wedge ((t' \notin \text{Actif}(M - \bullet t)) \vee t = t')$$

Intuitivement, ceci signifie que le tir d'une transition n'est pas considéré comme une étape atomique, mais qu'au contraire le système passe par un état intermédiaire. De plus, la transition tirée est toujours remise à zéro.

L'ensemble  $\text{ADM}(\mathcal{N})$  des *configurations (admissibles)* de  $\mathcal{N}$  est constitué des paires  $(M, \nu)$  telles que  $\nu(t) \in I(t)^\downarrow$  pour chaque transition  $t \in \text{Actif}(M)$ . Ainsi, le temps ne peut progresser dans une configuration admissible que s'il ne dépasse aucun des intervalles de tir des transitions actives. Cette condition forte d'urgence des réseaux de Petri temporels est une caractéristique essentielle de ce modèle. Elle implique en particulier une contrainte plus forte que l'urgence introduite par les invariants dans les automates temporisés : une transition ne peut pas être désactivée par écoulement du temps. Nous y reviendrons lors de la présentation des réseaux de Petri temporisés.

**Définition 2.7** (Sémantique d'un réseau de Petri temporel). *La sémantique d'un réseau de Petri temporel  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda, I)$  est le système de transitions temporisé  $\llbracket \mathcal{N} \rrbracket = (Q, q_0, \rightarrow)$  défini par :*

<sup>2</sup>Rappelons que  $\mathcal{I}$ , défini page 31, représente l'ensemble des intervalles à bornes rationnelles. Le modèle introduit initialement par Merlin impose en fait que ces intervalles soient fermés.

- $Q = ADM(\mathcal{N})$ ,
- $q_0 = (M_0, \mathbf{0})$  et
- $\rightarrow$  donnée par :
  - **transitions de délai** :  $(M, \nu) \xrightarrow{\tau} (M, \nu + \tau)$  ssi  $\forall t \in \text{Actif}(M)$ ,  $\nu(t) + \tau \in I(t)^\downarrow$ ,
  - **transitions discrètes** :  $(M, \nu) \xrightarrow{\Lambda(t)} (M - \bullet t + t^\bullet, \nu')$  ssi  $t \in \text{Actif}(M)$  est telle que  $\nu(t) \in I(t)$ , et  $\forall t' \in \text{Actif}(M - \bullet t + t^\bullet)$ ,  $\nu'(t') = \begin{cases} 0 & \text{si } \uparrow \text{Actif}(t', M, t) \\ \nu(t) & \text{sinon.} \end{cases}$

Considérons enfin une condition d'acceptation  $\text{Acc} = \{\text{acc}_1, \dots, \text{acc}_p\}$  de  $\mathcal{N}$ . Nous définissons alors l'ensemble des configurations finales de  $\llbracket \mathcal{N} \rrbracket$ , noté  $Q_f$ , par :

$$Q_f = \{(M, \nu) \in Q \mid \exists 1 \leq i \leq p \text{ tel que } M \text{ satisfait } \text{acc}_i\}$$

et la condition de Büchi généralisée associée à  $\llbracket \mathcal{N} \rrbracket$ , notée  $\{Q_r^1, \dots, Q_r^p\}$ , par :

$$Q_r^i = \{(M, \nu) \in Q \mid M \text{ satisfait } \text{acc}_i\}$$

Les définitions données pour les systèmes de transitions temporisés permettent donc d'utiliser les notions d'exécution temporisée, d'exécution temporisée acceptante, de mot temporisé accepté, et de langage temporisé accepté par un réseau de Petri temporel  $\mathcal{N}$ , noté  $\mathcal{L}(\mathcal{N})$ , identifié avec le langage temporisé accepté par son système de transitions temporisé  $\llbracket \mathcal{N} \rrbracket$ .

De plus, les notions de réseaux bornés,  $k$ -bornés et saufs s'étendent immédiatement aux réseaux de Petri temporels.

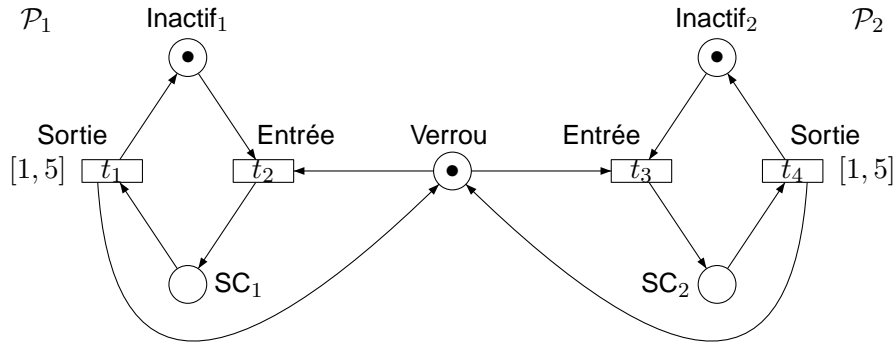


FIG. 2.3 – Exemple de réseau de Petri temporel modélisant l'exclusion mutuelle.

**Exemple 2.8** (Réseau de Petri temporel modélisant l'exclusion mutuelle). L'exemple considéré ici est représenté sur la figure 2.3 et consiste en une simple extension de l'exemple 2.5 dans laquelle nous avons ajouté des contraintes temporelles. Ainsi, une fois la section critique atteinte, un processus doit y rester au moins une unité de temps et doit la quitter au plus après 5 unités de temps

Plus formellement, le réseau de Petri temporel  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda, I)$  est donc donné par la même définition que plus haut, avec en plus :

$$I(t_1) = I(t_4) = [1, 5] \text{ et } I(t_2) = I(t_3) = [0, +\infty[$$

Notons que les intervalles  $[0, +\infty[$  ne sont pas représentés sur la figure 2.3. Nous utiliserons cette convention sur toutes les figures par la suite.

Afin d'introduire une priorité entre les processus pour privilégier celui qui n'a pas atteint la section critique au tour précédent, il est possible d'ajouter une temporisation dans le retour du jeton dans la

place Inactif. Ainsi, une place intermédiaire Inter permet d'imposer qu'au moins une unité de temps se soit écoulée avant que le processus puisse à nouveau atteindre sa section critique. Cette situation est représentée sur la figure 2.4. Avec cette nouvelle modélisation, quand un processus quitte sa section critique, l'autre processus est seul à pouvoir y accéder pendant au moins une unité de temps.  $\lrcorner$

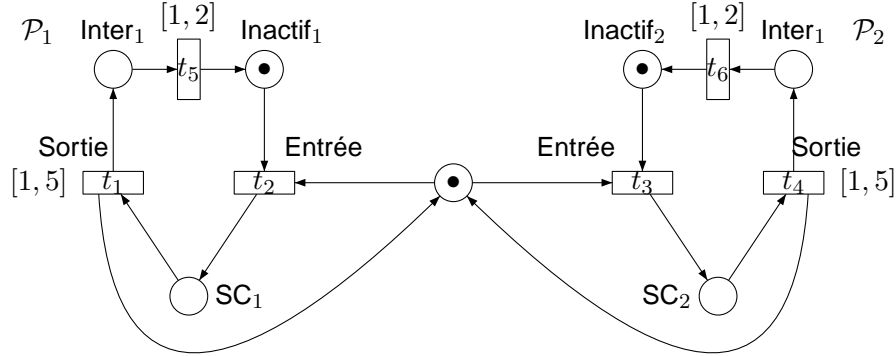


FIG. 2.4 – Exemple de réseau de Petri temporel modélisant l'exclusion mutuelle avec priorité.

## 2.2.2 Travaux existants

Nous mentionnons à présent les résultats principaux connus pour ce modèle. Les problèmes d'accessibilité, de couverture et de bornitude évoqués plus haut pour les réseaux de Petri s'étendent naturellement aux réseaux de Petri temporels. Cependant, la plupart de ces propriétés sont indécidables pour le modèle général, à commencer par la plus fondamentale, l'accessibilité d'un marquage discret. Ce problème consiste à décider s'il est possible d'atteindre une configuration dont seule la partie discrète est spécifiée, *i.e.* seulement son marquage et non la valuation de ses transitions actives.

**Théorème 2.9** ([JLL77]). *L'accessibilité est indécidable dans les réseaux de Petri temporels.*

**Preuve.** (Schéma) Nous présentons les idées mises en jeu dans cette preuve car elles permettent d'illustrer le fonctionnement du modèle. La preuve consiste en une réduction du problème de l'arrêt d'une machine de Minsky à deux compteurs [Min67]. Nous allons donc construire un réseau de Petri temporel permettant de simuler le comportement d'une machine à deux compteurs. Étant donnée une telle machine, ayant pour états  $q_1, \dots, q_n$ , et pour compteurs  $c_1$  et  $c_2$ , nous allons construire un réseau de Petri temporel ayant pour places  $q_1, \dots, q_n, c_1, c_2$  et dont les transitions sont représentées sur la figure 2.5 afin de coder les instructions de la machine de Minsky. Un unique jeton se trouve dans l'une des places  $q_1, \dots, q_n$  et représente l'état courant de la machine, tandis que le nombre de jetons présents dans la place  $c_i$  (pour  $i = 1, 2$ ) code la valeur du compteur  $c_i$ . L'opération consistant à incrémenter un compteur est très facile à réaliser, puisqu'il suffit de produire un jeton dans la place correspondante. En revanche, l'opération de décrémentation, représentée sur la sous-figure 2.5(b), est plus délicate. En effet, l'opération de test à zéro est impossible avec un réseau de Petri standard. L'ingrédient qui nous permet de coder cette opération avec les réseaux de Petri temporels est l'urgence du modèle sur laquelle nous avons insisté précédemment. Ainsi, le réseau doit d'abord tester s'il peut décrémentation le compteur (transition  $t_{-}$ ), *i.e.* s'il existe un jeton dans la place correspondante et seulement ensuite, il peut franchir la transition  $t_{=0}$  correspondant au cas du compteur nul. La priorité imposée entre ces deux transitions est obtenue grâce aux intervalles respectifs  $[0, 0]$  et  $[1, 1]$ . Ceci conclut la preuve du théorème.  $\square$

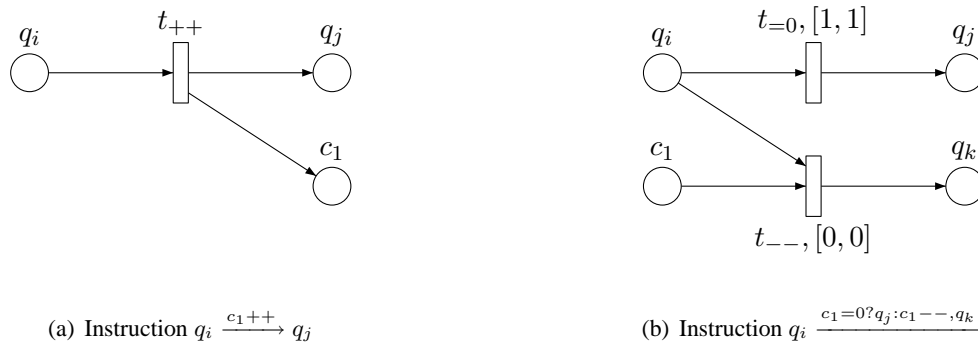


FIG. 2.5 – Codage des instructions d’une machine de Minsky à l’aide d’un réseau de Petri temporel.

Une adaptation simple de cette construction permet également de réduire l’arrêt des machines de Minsky à deux compteurs au problème de bornitude des réseaux de Petri temporels, démontrant ainsi que ce dernier problème est indécidable. De même, la construction précédente donne une preuve directe de l’indécidabilité du problème de couverture pour ce modèle (la couverture d’une place correspondant à un état de contrôle de la machine de Minsky est équivalente à son accessibilité). Pour les réseaux non bornés, les principaux problèmes sont donc tous indécidables.

En revanche, dans le cadre des réseaux de Petri temporels bornés, la plupart des problèmes classiques deviennent décidables, comme cela est énoncé dans le théorème suivant :

**Théorème 2.10.** *Étant donné un réseau de Petri temporel borné, les problèmes suivants sont décidables :*

- l’accessibilité et la couverture d’un marquage discret [BM83],
- la vérification de propriétés exprimées en logique temporelle linéaire (LTL) [BD91],
- la vérification de propriétés exprimées en logique temporelle arborescente (CTL) ou dans un fragment de la logique TCTL [BV03, TSLT97].

Ces résultats positifs de décidabilité ont été prouvés à l’aide d’une technique fondée sur la construction d’un graphe de simulation du réseau, appelé *graphe des classes*. Le graphe des classes, de la même façon que l’automate des régions introduit en sous-section 1.4.3 pour les automates temporisés, est une abstraction finie du réseau de Petri temporel. Il est fondé sur une autre sémantique des réseaux de Petri temporel. Cette nouvelle sémantique est équivalente à celle que nous avons présentée car les systèmes de transitions temporisés correspondants sont isomorphes. Celle-ci, au lieu de stocker dans la configuration le délai depuis lequel la transition est activée, la configuration enregistre une prédiction sur le futur, qui correspond à la quantité de temps restant à s’écouler avant le tir de la transition. Cette valeur est choisie de façon non déterministe dans l’intervalle de franchissement lorsqu’une transition devient active. Le graphe des classes est alors obtenu comme une représentation symbolique de cette sémantique. Plus précisément, dans ce graphe, chaque sommet est une paire constituée d’un marquage discret  $M$  et d’une contrainte d’horloges  $Z$  portant sur les horloges  $x_t$ , où  $t$  correspond à une transition activée, et sur une horloge supplémentaire  $x_0$  dont la valeur est toujours 0. La valeur de l’horloge  $x_t$  représente le délai avant le prochain tir de  $t$ .

Pour plus de détails sur cette construction, le lecteur peut se référer à [BM83]. Plusieurs extensions de cette construction ont été développées afin de vérifier des propriétés exprimées dans la logique temporelle linéaire ou la logique arborescente. De plus, ces techniques ont donné lieu au développement d’un outil, nommé TINA [BRV04], dédié à l’analyse des réseaux de Petri, ainsi qu’à l’analyse des réseaux de Petri temporels.

Un autre outil, nommé ROMEO [GLMR05], a été développé pour l'analyse des réseaux de Petri temporels. Celui-ci implémente deux approches. La première, présentée dans [LR03], consiste à traduire le réseau de Petri temporel en un automate temporisé qui lui est bisimilaire puis à utiliser les outils existants pour le modèle des automates temporisés. La seconde technique construit un graphe des zones du réseau de Petri temporel obtenu en attachant des zones au graphe des marquages accessibles [GRR03]. Nous présenterons dans le chapitre 4 une traduction des réseaux d'automates temporisés vers les réseaux de Petri temporels.

## 2.3 Réseaux de Petri temporisés

La seconde extension temporisée des réseaux de Petri que nous présentons, appelée réseaux de Petri temporisés, a été introduite dans [BLT90]. Le temps est cette fois introduit par le biais des âges des jetons.

### 2.3.1 Définition

**Définition 2.11** (Réseau de Petri temporisé). *Un réseau de Petri temporisé  $\mathcal{N}$  sur  $\Sigma_\varepsilon$  est un uplet  $(P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  où :*

- $P$  est un ensemble fini de places,
- $T$  est un ensemble fini de transitions avec  $P \cap T = \emptyset$ ,
- $\bullet(\cdot) \in \text{MEns}(P \times \mathcal{I})^T$  est l'application d'incidence arrière,
- $(\cdot)^\bullet \in \text{MEns}(P \times \mathcal{I})^T$  est l'application d'incidence avant,
- $M_0 \in \text{MEns}(P)$  est le marquage initial,
- $\Lambda : P \rightarrow \Sigma_\varepsilon$  est la fonction d'étiquetage.

Dans ce modèle, le temps est introduit au niveau des jetons. Une configuration d'un réseau de Petri temporisé sera donc la donnée d'un marquage discret, comme précédemment, ainsi que d'une valeur temporelle pour chacun des jetons. L'objet mathématique correspondant est donc une application de l'ensemble des places vers les multi-ensembles d'éléments de  $\mathbb{T}$ , *i.e.* un élément de  $\text{MEns}(\mathbb{T})^P$ , ou de façon équivalente de  $\text{MEns}(P \times \mathbb{T})$ . Nous représenterons un jeton situé dans la place  $p$  et d'âge  $x$  avec la notation  $(p, x)$ . Une configuration est donc représentée par une somme finie de telles paires. En particulier, un jeton  $(p, x)$  appartient à une configuration  $\nu$  si et seulement si  $(p, x) \leq \nu$ . La *configuration initiale*  $\nu_0 \in \text{MEns}(P \times \mathbb{T})$  est définie par  $\nu_0 = \sum_{p \in P} M_0(p) \cdot (p, 0)$  (il y a  $M_0(p)$  jetons d'âge 0 dans la place  $p$ ).

Avant de présenter la sémantique des réseaux de Petri temporisés, nous introduisons une relation de satisfaction. Étant donné une configuration d'un tel réseau, *i.e.* un élément  $\nu \in \text{MEns}(P \times \mathbb{T})$ , et l'étiquette d'un arc d'un tel réseau, *i.e.* un élément  $f \in \text{MEns}(P \times \mathcal{I})$ , nous disons que  $\nu$  satisfait  $f$ , noté  $\nu \models f$ , si et seulement si il existe  $x \in \text{MEns}(P \times \mathbb{T} \times \mathcal{I})$  vérifiant les propriétés suivantes :

$$\begin{cases} \pi_{1,2}(x) = \nu, \\ \pi_{1,3}(x) = f, \\ \forall (p, \tau, I) \in \text{dom}(x), \tau \in I. \end{cases}$$

Pour qu'une transition  $t$  soit *franchissable* (ou *tirable*), il faut trouver des jetons dans les places en entrée de la transition, comme pour un réseau de Petri, et il faut également s'assurer pour chaque jeton sélectionné que son âge appartient à l'intervalle correspondant indiqué par  $\bullet t$ . Lors du tir de cette transition, les jetons précédents sont consommés et donc retirés de la configuration courante. De

nouveaux jetons sont produits dans les places indiquées par  $t^\bullet$ . L'âge affecté à ces jetons est choisi de façon non déterministe dans les intervalles correspondants donnés par  $t^\bullet$ .

Nous donnons à présent la sémantique d'un réseau de Petri temporisé en termes de système de transitions temporisé.

**Définition 2.12** (Sémantique d'un réseau de Petri temporisé). *Soit  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  un réseau de Petri temporisé. Sa sémantique est définie par le système de transitions temporisé  $\llbracket \mathcal{N} \rrbracket = (Q, q_0, \rightarrow)$  avec :*

- $Q = \text{MEns}(P \times \mathbb{T})$ ,
- $q_0 = \nu_0$ ,
- $\rightarrow$  définie pour  $\nu \in \text{MEns}(P \times \mathbb{T})$  par :
  - **transitions de délai** : pour  $d \in \mathbb{T}$ ,  $\nu \xrightarrow{d} \nu + d$ , où la somme est prise sur la composante de  $\mathbb{T}$ , i.e.  $(\nu + d)(p) = \nu(p) + d$  pour toute place  $p \in P$ .
  - **transitions discrètes** : pour  $a \in \Sigma_\varepsilon$ ,  $\nu \xrightarrow{a} \nu'$  si et seulement s'il existe un élément  $t \in T$  et des éléments  $\bullet\nu, \nu^\bullet \in \text{MEns}(P \times \mathbb{R}_{\geq 0})$  tels que :

$$\left\{ \begin{array}{l} \Lambda(t) = a \\ \bullet\nu \models \bullet t \\ \nu^\bullet \models t^\bullet \\ \bullet\nu \leq \nu \\ \nu' = \nu - \bullet\nu + \nu^\bullet \end{array} \right.$$

Enfin, étant donnée une condition d'acceptation  $\text{ACC} = \{\text{acc}_1, \dots, \text{acc}_p\}$ , nous définissons les configurations finales  $Q_f$  de  $\llbracket \mathcal{N} \rrbracket$  ainsi :

$$Q_f = \{\nu \in \text{MEns}(P \times \mathbb{T}) \mid \exists 1 \leq i \leq p \text{ tel que } \pi_1(\nu) \text{ satisfait } \text{acc}_i\}$$

Pour les exécutions infinies, nous interprétons  $\text{ACC}$  comme une condition d'acceptation de Büchi généralisée. La condition de Büchi généralisée  $\{Q_r^1, \dots, Q_r^p\}$  du système de transitions temporisé  $\llbracket \mathcal{N} \rrbracket$  est alors définie par

$$Q_r^i = \{\nu \in \text{MEns}(P \times \mathbb{T}) \mid \pi_1(\nu) \text{ satisfait } \text{acc}_i\}$$

pour tout indice  $i$  compris entre 1 et  $p$ .

Une fois ces définitions établies, nous pouvons utiliser les notions d'exécution temporisée, d'exécution temporisée acceptante, de mot temporisé accepté et de langage temporisé accepté par un réseau de Petri temporisé  $\mathcal{N}$  en l'identifiant avec son système de transitions temporisé  $\llbracket \mathcal{N} \rrbracket$ .

Il est important de remarquer que dans ce modèle, le temps peut s'écouler de façon arbitraire. En particulier, il n'y a pas d'urgence et une transition tirable peut ne jamais être franchie. Ceci implique que des jetons peuvent atteindre un âge qui leur interdit d'être consommé par la suite, ce sont des *jetons morts*.

**Sous-classes des réseaux de Petri temporisés.** En plus des restrictions sémantiques présentées pour les réseaux de Petri ( $k$ -borné, borné et sauf), nous distinguons des restrictions syntaxiques du modèle présenté précédemment que nous serons amenés à considérer par la suite. Nous considérons un réseau de Petri temporisé  $\mathcal{N}$ , et dirons que  $\mathcal{N}$  est :

- *intégral* si tous les intervalles apparaissant dans sa définition sont des éléments de  $\mathcal{I}_{\mathbb{N}}$ ,

- à remises à zéro si l'unique intervalle autorisé dans la définition de la fonction  $(.)^\bullet$  est l'intervalle  $[0, 0]$ , i.e.  $I \neq [0, 0] \Rightarrow \forall p \in P, I \notin t^\bullet(p)$ .

**Exemple 2.13** (Réseau de Petri temporisé modélisant l'exclusion mutuelle). L'exemple donné ici et représenté sur la figure 2.6 est à nouveau une variation des exemples précédents. Nous modélisons cette fois l'exclusion mutuelle de deux processus à l'aide d'un réseau de Petri temporisé. Nous signalons quelques conventions dans la représentation de ces objets. Étant données une place  $p$  et une transition  $t$ ,

- en consommation, l'intervalle par défaut est  $[0, +\infty[$ , et ne sera alors pas indiqué,
- en production, l'intervalle par défaut est  $[0, 0]$ , et ne sera alors pas indiqué.

Ainsi, sur l'exemple représenté sur la figure 2.6, nous détaillons l'exemple de la transition  $t_2$  étiquetée par Entrée dans le processus  $\mathcal{P}_1$  :

- $\bullet t_2(\text{Verrou}) = \bullet t_2(\text{Inactif}_1) = [0, +\infty[$
- $t_2^\bullet(\text{SC}_1) = [0, 0]$

puis le cas de la transition  $t_1$  :

- $\bullet t_1(\text{SC}_1) = [1, 5]$
- $t_1^\bullet(\text{Inactif}_1) = t_1^\bullet(\text{Verrou}) = [0, 0]$

Le système représenté est similaire à celui des exemples précédents, mais les comportements autorisés ne sont en fait pas les mêmes, du fait de l'absence d'urgence dans la sémantique des réseaux de Petri temporisés. En effet, des blocages peuvent se produire si par exemple un jeton n'est pas consommé dans une des places  $\text{SC}_1$  et  $\text{SC}_2$ . Afin de pallier ce défaut, nous pouvons ajouter la condition d'acceptation suivante :  $\text{SC}_1 + \text{SC}_2 \leq 0$ . Les comportements acceptés par le réseau de Petri temporisés possèdent alors la vivacité présente dans les réseaux de Petri temporels grâce à l'urgence. Cependant, ceci ne concerne que les comportements acceptés et pas l'ensemble des comportements du système.

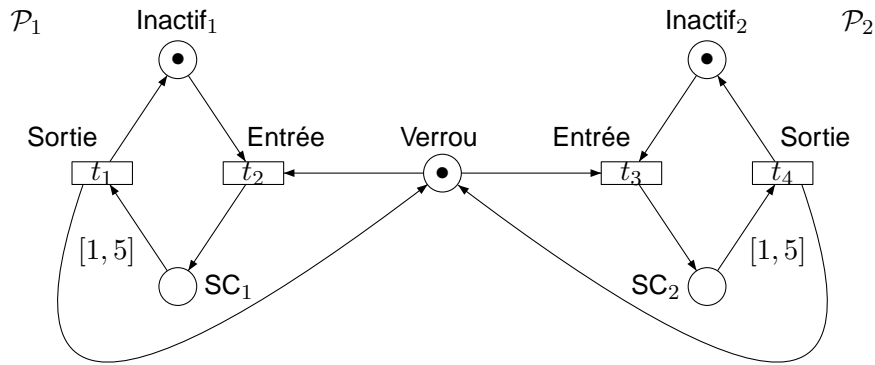


FIG. 2.6 – Exemple de réseau de Petri temporisé modélisant l'exclusion mutuelle.

┘

### 2.3.2 Travaux existants

Comme mentionné précédemment, la propriété d'urgence mentionnée présente dans les réseaux de Petri temporels n'existe pas dans le modèle des réseaux de Petri temporisés. Ceci a des conséquences



importantes sur les résultats de décidabilité. En effet, sans cette condition d'urgence, une transition franchissable depuis un marquage (temporisé) donné l'est également depuis n'importe quel marquage plus grand, *i.e.* contenant davantage de jetons. Ceci implique une analyse plus facile du modèle. Ainsi, plusieurs problèmes ont été prouvés décidables pour cette classe.

**Théorème 2.14** ([dFERA00, AMM07]). *Les problèmes suivants sont décidables pour le modèle des réseaux de Petri temporisés :*

- le problème de couverture,
- la Zenoness, *i.e.* l'existence d'une exécution infinie de durée finie,
- la vivacité des jetons, *i.e.* étant donné un marquage et un jeton de ce marquage, déterminer s'il existe une exécution qui consommera ce jeton,
- la bornitude, en prenant en compte les jetons morts.

En revanche, cette remarque n'est pas suffisante pour obtenir la décidabilité de l'accessibilité et ce problème demeure indécidable.

**Théorème 2.15** ([VFEC99]). *Le problème de l'accessibilité est indécidable pour le modèle des réseaux de Petri temporisés.*

**Preuve.** (Schéma) La preuve procède à nouveau par réduction du problème de l'arrêt des machines de Minsky à deux compteurs. Cependant, sans la condition d'urgence, il n'est plus possible d'imposer directement lors de l'exécution une priorité entre deux transitions. Afin de pallier cette difficulté, une condition d'acceptation plus stricte requiert que certaines places soient vides à l'issue de l'exécution. Cette condition a posteriori impose certains franchissements au cours de l'exécution.

Plus précisément, une première étape consiste à remarquer qu'il est possible d'imposer la condition supplémentaire suivante à la machine à deux compteurs simulée : une fois l'état cible atteint, la machine possède des instructions lui permettant de remettre à zéro ses compteurs. Ainsi, nous cherchons alors à décider s'il existe une exécution de la machine atteignant un certain état cible dans une configuration dans laquelle ses deux compteurs sont nuls. Ce nouveau problème est encore indécidable. Le codage de la machine utilisé pour cette réduction est similaire à celui utilisé dans la preuve du Théorème 2.9. Les instructions d'incrément et de décrémentation sont alors modélisées par les réseaux représentés sur la figure 2.7 (rappelons que les arcs de production sont étiquetés par défaut par l'intervalle  $[0, 0]$ ). L'incrément est naturelle, il suffit de remarquer qu'en plus, la transition  $t_{++}$  doit être franchie immédiatement. Pour la décrémentation, l'idée initiale est la même : la transition correspondant au cas du compteur nul ( $t_{=0}$ ) ne peut être franchie qu'après (au sens temporel) celle ( $t_{--}$ ) correspondant au cas du compteur strictement positif (intervalle  $[1, 1]$  contre  $[0, 0]$ ). De plus, nous modifions la condition d'acceptation en ajoutant la contrainte  $c_1 + c_2 \leq 0$ . Comme la sémantique n'est pas urgente, il existe des exécutions dans lesquelles le réseau tire la transition  $t_{=0}$ , même si  $t_{--}$  était franchissable. Dans ce cas, l'âge des jetons présents dans la place  $c_1$  a augmenté de 1, et ceux-ci deviennent alors des jetons morts. En particulier, ils ne pourront plus être consommés, et l'exécution ne pourra plus atteindre une configuration acceptante, *i.e.* vérifiant la condition  $c_1 + c_2 \leq 0$ . Ainsi, si la simulation n'est pas « honnête », elle ne sera pas acceptée. En revanche, si la transition  $t_{=0}$  est franchie exactement quand la place  $c_1$  est vide, alors la transition  $t_{raz}$  lui permet de remettre à zéro les âges des jetons éventuellement présents dans la place  $c_2$ . Ainsi, la simulation permet de représenter la valeur des compteurs par le nombre de jetons présents dans la place correspondante, et vérifiant de plus le fait que tous ces jetons sont d'âge nul.

Finalement, toute exécution acceptante simule bien le comportement de la machine à deux compteurs, mais il n'est pas possible de contraindre les exécutions non acceptantes. Nous réutiliserons cette technique de preuve dans le chapitre 5. □

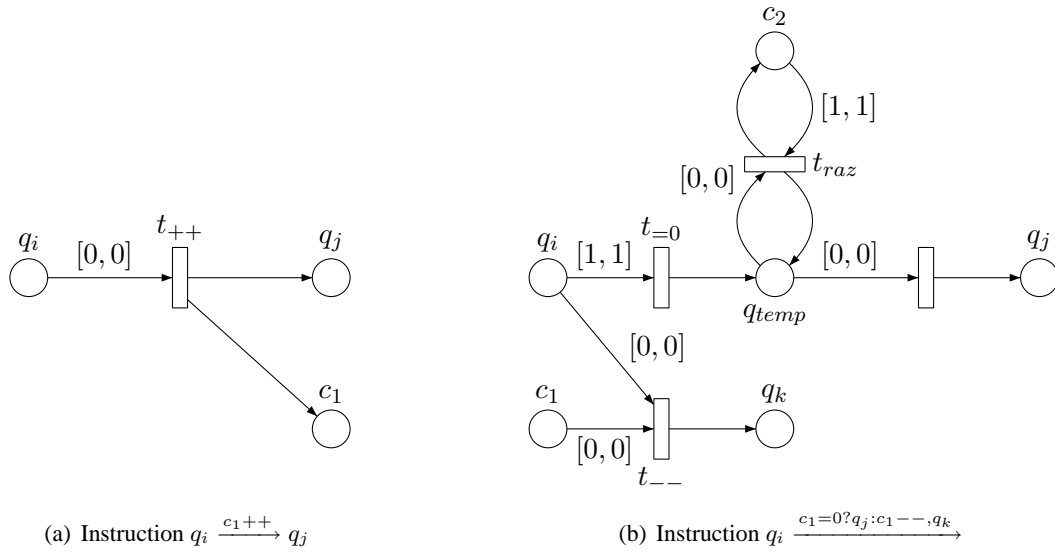


FIG. 2.7 – Codage des instructions d’une machine de Minsky à l’aide d’un réseau de Petri temporisé.

Enfin, comme pour le modèle précédent, le cadre des réseaux bornés est plus favorable et les problèmes majeurs y sont décidables.

**Théorème 2.16.** *Étant donné un réseau de Petri temporisé borné, les problèmes suivants sont décidables :*

- l’accessibilité d’un marquage discret,
- la vérification de propriétés exprimées en logique temporelle linéaire.

## 2.4 Conclusion

Nous avons déjà insisté au cours de la présentation des deux modèles sur l’importance de la notion d’urgence, présente dans la sémantique des réseaux de Petri temporels, mais absente dans les réseaux de Petri temporisés. Cette notion d’urgence permet d’ajouter une importante vivacité au modèle, *i.e.* d’imposer à un réseau de Petri temporel de franchir des transitions. Cette propriété est à rapprocher du rôle des invariants dans les automates temporels. Cependant, dans un réseau de Petri temporel, toutes les transitions ont une sémantique urgente. Au contraire, dans un réseau de Petri temporisé, aucune transition n’est urgente. Une comparaison des pouvoirs d’expression en termes de bisimilarité de différentes extensions temporisées des réseaux de Petri peut être trouvée dans [BR07].

Dans les réseaux de Petri temporisés, nous avons montré dans la preuve de l’indécidabilité de l’accessibilité comment les conditions d’acceptation permettent de compenser l’absence d’urgence dans la sémantique. Ainsi, les comportements acceptés vont contenir une certaine vivacité, mais les comportements dans leur ensemble seront moins contraints. Cette remarque s’illustre bien sur les exemples 2.8 et 2.13 donnés précédemment.

Une seconde conséquence de la présence (ou de l’absence) d’urgence dans le modèle est l’impact sur la décidabilité du modèle. Ainsi, l’absence d’urgence apporte de la régularité aux conditions de franchissement et une plus forte décidabilité du modèle des réseaux de Petri temporisés. Nous avons vu que le problème de couverture, entre autres, y est décidable.

## **Deuxième partie**

# **Propriétés et Comparaison de Modèles Temporisés**



## Chapitre 3

# Résultats d'indécidabilité pour les automates temporisés avec transitions silencieuses

Dans ce chapitre, nous nous intéressons à la classe des automates temporisés étendus avec des transitions silencieuses et démontrons une série de résultats d'indécidabilité pour des problèmes naturels relatifs à cette classe. Les résultats présentés ici sont issus de [BHR07].

### 3.1 Introduction

Nous avons défini dans le chapitre 1 le modèle des automates temporisés. Nous avons également introduit dans la sous-section 1.5.1 son extension avec des transitions silencieuses et mentionné certains résultats fondamentaux la concernant.

La compréhension théorique du modèle des automates temporisés, ainsi que les fondements théoriques des langages temporisés ont toujours suscité un intérêt important. En effet les langages formels standards (non temporisés) possèdent des propriétés intéressantes. Ainsi, les langages réguliers peuvent être caractérisés de différentes manières : expressions rationnelles, théorème de Kleene, reconnaissance par monoïdes, logique monadique du second ordre. Ce type d'équivalence existe également pour des sous-classes de ces langages (logique du premier ordre et langages réguliers apériodiques), et l'ensemble de ces caractérisations constitue un cadre bien établi et uniformément reconnu.

Le cas des langages temporisés est nettement moins satisfaisant puisqu'ils ne possèdent pas ce type de caractérisation logique et/ou algébrique, bien que ce sujet ait été abordé de différentes manières (voir par exemple [Wil94, Dim99, D'S00, Dim01, ACM02, BP02, BPT02, MP04, CDP06]). En effet, la « bonne » classe de langages temporisés n'a sans doute pas encore été identifiée. Un travail important doit donc encore être mené afin de mieux comprendre et formaliser les fondements théoriques des langages temporisés [Asa04].

Un défaut majeur des automates temporisés (et des langages temporisés qu'ils acceptent) est qu'ils ne sont pas clos par complémentation et ne sont pas déterminisables. Ceci rend plus difficile une définition équivalente en termes de logiques puisque la fermeture par négation constitue d'une certaine façon la quintessence des logiques. Par conséquent, nous devons ou bien abandonner la négation dans les logiques [Wil94, Bou02], ou bien nous restreindre à une sous-classe des langages fermée par complémentation [AFH94, D'S00, CDP06], ou encore essayer de mieux comprendre le rôle de la complémentation. Les travaux présentés dans [Tri03] suivent cette dernière idée et soulèvent des

questions comme « un automate temporisé est-il complémentable ? » ou « un automate temporisé est-il déterminisable ? ». Dans ces travaux, Tripakis démontre que ces deux problèmes sont indécidables si en plus l'automate démontrant la propriété doit être synthétisé. D'autres problèmes sont démontrés indécidables avec cette même condition comme la minimisation du nombre d'horloges, *etc.* Plus récemment, Finkel a amélioré significativement ces résultats en démontrant dans [Fin06] que tous ces problèmes sont indécidables même sans exiger la construction de l'automate « témoin ».

Dans le contexte non temporisé, les transitions silencieuses n'augmentent pas le pouvoir expressif du modèle des automates finis. Pour les systèmes temporisés, nous avons vu (théorème 1.21) que cela n'est pas le cas. Pourtant, du point de vue de la vérification, ces transitions peuvent être utiles afin de représenter des actions non observables du système ou encore pour modéliser des comportements discrets au sein d'un environnement continu. De plus, elles n'induisent aucune complexité dans les techniques classiques de vérification comme le graphe des régions (sous-section 1.4.3) ou les algorithmes fondés sur les zones (section 6.2).

Dans ce chapitre, nous poursuivons simultanément les deux travaux [BDGP98, Fin06]. D'abord, nous répondons négativement à une question centrale issue de l'introduction des transitions silencieuses : « Est-il possible de déterminer si le langage d'un automate temporisé avec transitions silencieuses peut être accepté par un automate temporisé sans transitions silencieuses ? » Nous étendons ensuite chacun des résultats de [Fin06] au cadre des automates temporisés avec transitions silencieuses. Bien que nos preuves suivent les mêmes lignes, elles sont nettement plus délicates car toutes les preuves de [Fin06] s'appuient sur le fait que dans un automate temporisé *sans* transitions silencieuses, étant donné un mot temporisé, il existe un nombre fini d'exécutions acceptant ce mot. Ce n'est plus le cas en présence de transitions silencieuses puisque ces exécutions peuvent même être en quantité non dénombrable.

Plus précisément, nous démontrons dans ce chapitre qu'il est impossible de :

- décider si un langage temporisé  $\varepsilon$ -régulier est simplement régulier (*i.e.* s'il est possible de s'affranchir des transitions silencieuses dans l'automate l'acceptant), *cf* section 3.3 ;
- décider si le complément d'un langage temporisé  $\varepsilon$ -régulier est également  $\varepsilon$ -régulier, *cf* section 3.4 ;
- calculer le nombre minimal d'horloges nécessaires pour reconnaître un langage temporisé  $\varepsilon$ -régulier, *cf* section 3.5 ;
- décider si le mélange de deux langages temporisés ( $\varepsilon$ -)réguliers est un langage  $\varepsilon$ -régulier, *cf* section 3.6.

Enfin, nous étendons tous ces précédents résultats, démontrés pour des langages de mots finis, au cadre des mots infinis et des automates temporisés équipés de conditions d'acceptation de Büchi, *cf* section 3.7.

## 3.2 Préliminaires

Nous commençons dans cette première partie par rappeler certains préliminaires nécessaires dans la suite de ce chapitre.

**Notations.** Considérons une exécution (temporisée)  $\rho$  de  $\mathcal{A}$ . Cette exécution s'écrit  $\rho = (\ell_0, v_0) \xrightarrow{d_0} (\ell_0, v_0 + d_0) \xrightarrow{a_0} (\ell_1, v_1) \xrightarrow{d_1} (\ell_1, v_1 + d_1) \xrightarrow{a_1} \dots$ . Si  $\tau \in \mathbb{T}$  est une valeur inférieure ou égale à la durée d'une exécution finie  $\rho$ , nous écrivons  $(\ell_{\rho, \tau}^-, v_{\rho, \tau}^-)$  pour la première configuration le long de  $\rho$  atteinte par l'automate à la date  $\tau$ , et  $(\ell_{\rho, \tau}^+, v_{\rho, \tau}^+)$  pour la dernière configuration atteinte à la date  $\tau$ .

Plus formellement, définissons  $i^- = \max\{i \mid \tau_i < \tau\}$  et  $i^+ = \max\{i \mid \tau_i \leq \tau\}$  avec la convention que  $\max(\emptyset) = 0$ , nous avons alors :

$$\begin{cases} (\ell_{\varrho, \tau}^-, v_{\varrho, \tau}^-) = (\ell_{i^-}, v_{i^-} + \tau - \tau_{i^-}), \text{ et} \\ (\ell_{\varrho, \tau}^+, v_{\varrho, \tau}^+) = (\ell_{i^+}, v_{i^+} + \tau - \tau_{i^+}). \end{cases}$$

Par exemple, si une unique transition discrète intervient à la date  $\tau$ , alors  $(\ell_{\varrho, \tau}^-, v_{\varrho, \tau}^-)$  est la configuration juste avant le franchissement de cette transition, tandis que  $(\ell_{\varrho, \tau}^+, v_{\varrho, \tau}^+)$  est la configuration atteinte à l'issue de ce franchissement. Dans notre modèle, nous autorisons le franchissement de plusieurs transitions discrètes à une même date  $\tau$ . Pour de telles séquences de transitions « instantanées », l'exécution atteint de nombreuses configurations à la date  $\tau$ . Les deux configurations que nous avons mises en évidence correspondent respectivement à la première et à la dernière de ces configurations.

D'une manière générale, si  $\tau > 0$  alors la configuration  $(\ell_{\varrho, \tau}^-, v_{\varrho, \tau}^-)$  est toujours atteinte après un écoulement de temps et vérifie en particulier  $\forall x \in X, v_{\varrho, \tau}^-(x) > 0$  (si  $X$  représente l'ensemble d'horloges).

**Résultats classiques sur les automates temporisés.** Nous rassemblons ici les résultats d'expressivité et de décidabilité (ou indécidabilité) dont nous aurons besoin dans nos preuves. La plupart d'entre eux sont standards et nous donnons pour chacun une référence.

**Théorème 3.1** (Résultats de clôture et d'expressivité).

1. La famille des langages temporisés réguliers est strictement incluse dans la famille des langages temporisés  $\varepsilon$ -réguliers ( [BDGP98], voir aussi théorème 1.21).
2. La famille des langages temporisés réguliers n'est pas close par complémentation [AD94].
3. La famille des langages temporisés réguliers acceptés par un automate temporisé déterministe est strictement incluse dans la famille des langages temporisés réguliers [AD94].

**Théorème 3.2** (Problème de l'universalité).

1. Le problème de l'universalité est indécidable pour les automates temporisés ayant deux horloges ou plus [AD94].
2. Le problème de l'universalité est décidable pour les automates temporisés à une horloge [OW04].
3. Le problème de l'universalité est indécidable pour les automates temporisés avec transitions silencieuses à une horloge [LW07].

**Exécutions temporisées dans les automates temporisés.** Dans cette sous-section, nous donnons deux exemples importants d'automates temporisés avec transitions silencieuses. Le premier sera réutilisé par la suite et le second permet d'illustrer les notations introduites précédemment.

**Exemple 3.3.** L'automate temporisé avec transitions silencieuses représenté sur la figure 3.1 accepte le langage temporisé

$$\mathcal{R}_{\text{pair}} = \{(a, \tau_1)(a, \tau_2) \dots (a, \tau_n) \mid \tau_i \equiv 0 \pmod{2} \text{ pour tout } 1 \leq i \leq n\}.$$

Il est démontré dans [BDGP98] que ce langage temporisé n'est accepté par aucun automate temporisé sans transitions silencieuses, illustrant ainsi le résultat (1) du théorème 3.1. ┘

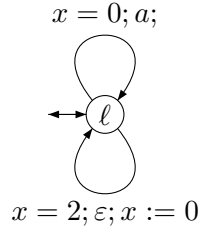
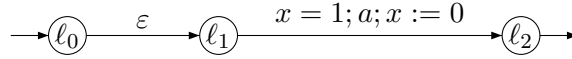


FIG. 3.1 – Un automate temporisé avec transitions silencieuses équivalent à aucun automate temporisé

**Exemple 3.4.** L'automate temporisé avec transitions silencieuses représenté sur la figure 3.2 accepte le langage temporisé réduit au singleton  $\{(a, 1)\}$ . Pourtant, toute exécution  $(\ell_0, 0) \xrightarrow{d} (\ell_0, d) \xrightarrow{\varepsilon} (\ell_1, d) \xrightarrow{1-d} (\ell_1, 1) \xrightarrow{a} (\ell_2, 1)$ , avec  $d \in [0, 1]$ , est une exécution acceptant ce mot temporisé. Le long d'une de ces exécutions, disons  $\varrho$ , les configurations  $(\ell_{\varrho,1}^-, v_{\varrho,1}^-)$  et  $(\ell_{\varrho,1}^+, v_{\varrho,1}^+)$  vérifient  $(\ell_{\varrho,1}^-, v_{\varrho,1}^-) = (\ell_1, 1)$  et  $(\ell_{\varrho,1}^+, v_{\varrho,1}^+) = (\ell_2, 0)$ .  $\lrcorner$

FIG. 3.2 –  $(a, 1)$  est accepté par une quantité non dénombrable d'exécutions

### 3.3 S'affranchir des transitions silencieuses

L'impact des transitions silencieuses a été étudié dans [BDGP98] et des restrictions syntaxiques ont été obtenues qui permettent de garantir qu'il est possible de s'affranchir des transitions silencieuses. En d'autres termes, ces restrictions syntaxiques faites sur un automate temporisé avec transitions silencieuses assurent qu'il accepte un langage régulier. Cependant, ces restrictions ne sont pas nécessaires et nous démontrons dans cette section que le problème consistant à déterminer si un langage temporisé  $\varepsilon$ -régulier est un langage régulier, *i.e.*, étant donné un automate temporisé avec transitions silencieuses, s'il est possible de s'affranchir de celles-ci, est un problème indécidable.

**Théorème 3.5** (Retirer les transitions silencieuses). *Étant donné un automate temporisé avec transitions silencieuses  $\mathcal{A}$ , il est indécidable de déterminer s'il existe un automate temporisé  $\mathcal{B}$  tel que  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ .*

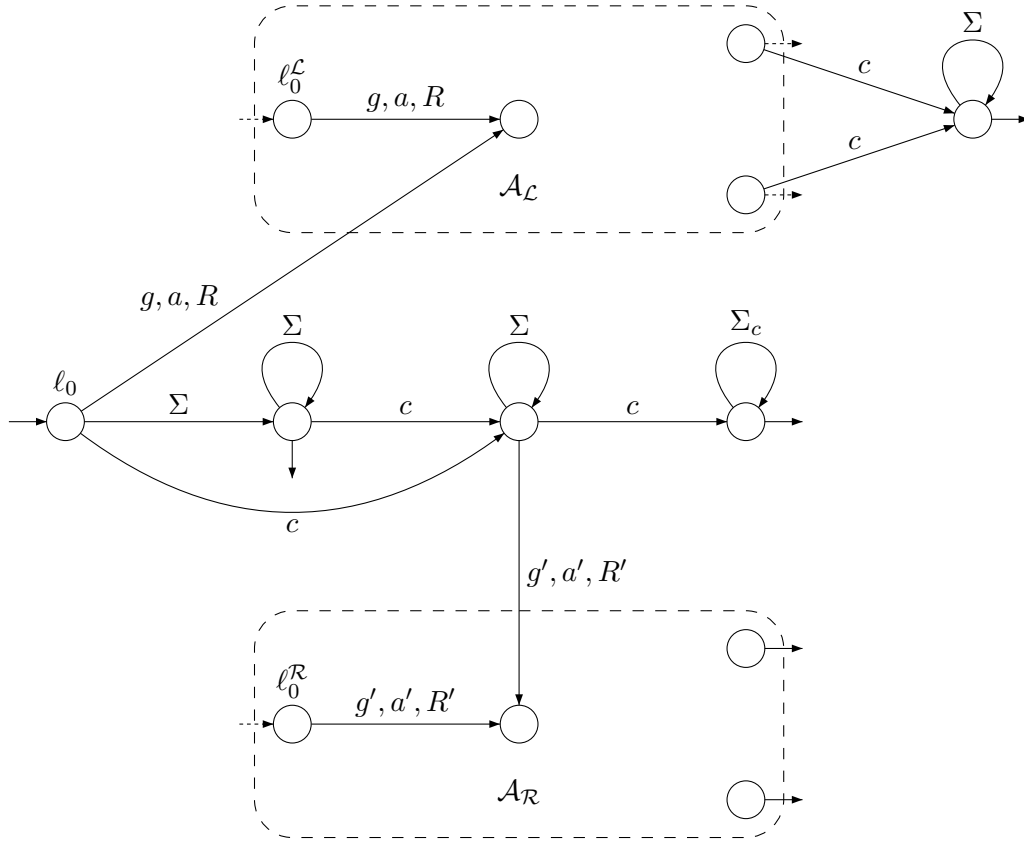
Afin de démontrer ce résultat, mais aussi la plupart des autres résultats d'indécidabilité de ce chapitre, nous réduisons le problème de l'universalité des automates temporisés au problème que nous étudions. Nous décrivons d'abord une construction portant sur les langages temporisés introduite dans [Fin06].

Dans la suite,  $\Sigma$  représente un alphabet, et  $c$  une nouvelle lettre n'appartenant pas à  $\Sigma$ . Nous notons  $\Sigma_c = \Sigma \cup \{c\}$ .

**Définition 3.6.** *Soient  $\mathcal{L}$  et  $\mathcal{R}$  deux langages temporisés sur  $\Sigma$ . Alors  $\text{Compose}(\mathcal{L}, \mathcal{R})$  est le langage temporisé sur  $\Sigma_c$  défini comme l'union des trois langages suivants :*

$$\begin{aligned} \mathcal{V}_1 &= \{w \in \mathcal{MT}^*(\Sigma_c) \mid \exists w' \in \mathcal{L}, \exists w'' \in \mathcal{MT}^*(\Sigma), \exists \tau \text{ t.q. } w = w'(c, \tau)w''\} \\ \mathcal{V}_2 &= \{w \in \mathcal{MT}^*(\Sigma_c) \mid |w|_c \neq 1\} \\ \mathcal{V}_3 &= \{w \in \mathcal{MT}^*(\Sigma_c) \mid \exists w' \in \mathcal{MT}^*(\Sigma), \exists w'' \in \mathcal{R}, \exists \tau \text{ t.q. } w = w'(c, \tau)(w'' + \tau)\} \end{aligned}$$



FIG. 3.3 – Automate reconnaissant le langage  $Compose(\mathcal{L}, \mathcal{R})$ .

Nous établissons à présent deux propriétés importantes de cette construction que nous utiliserons par la suite.

**Lemme 3.7.** Soit  $\mathcal{L}$  et  $\mathcal{R}$  deux langages temporisés sur l'alphabet  $\Sigma$ .

- si  $\mathcal{L}$  et  $\mathcal{R}$  sont acceptés par des automates temporisés (respectivement avec transitions silencieuses) ayant moins de  $n$  horloges, alors  $Compose(\mathcal{L}, \mathcal{R})$  est également accepté par un automate temporisé (respectivement avec transitions silencieuses) ayant moins de  $n$  horloges.
- $Compose(\mathcal{M}T^*(\Sigma), \mathcal{R}) = \mathcal{M}T^*(\Sigma_c)$ , il est donc en particulier accepté par un automate temporisé déterministe sans horloges.

**Preuve.** Le premier point est obtenu en combinant les deux automates temporisés. La figure 3.3 représente cette construction. De plus, nous vérifions que les deux « sous automates »  $\mathcal{A}_{\mathcal{L}}$  et  $\mathcal{A}_{\mathcal{R}}$  peuvent partager leurs horloges et que le nombre d'horloges n'augmente donc pas lors de cette construction. Le second point est une simple conséquence de la définition.  $\square$

Nous pouvons à présent démontrer le théorème 3.5.

**Preuve.** Supposons que la lettre  $a$  appartienne à  $\Sigma$  et considérons le langage  $\mathcal{R}_{pair}$  introduit dans la section 3.2. Comme rappelé précédemment,  $\mathcal{R}_{pair}$  est  $\varepsilon$ -régulier, mais n'est pas régulier. Soit  $\mathcal{L} \subseteq \mathcal{M}T^*(\Sigma)$  un langage temporisé régulier. Considérons à présent le langage temporisé  $\mathcal{V} = Compose(\mathcal{L}, \mathcal{R}_{pair})$ . Appliquant le lemme 3.7, nous savons que  $\mathcal{V}$  est  $\varepsilon$ -régulier. Nous allons montrer que  $\mathcal{V}$  est régulier si et seulement si  $\mathcal{L}$  est universel sur  $\Sigma$ . Nous distinguons deux cas :

- (1) **Premier cas.** Supposons que  $\mathcal{L} = \mathcal{MT}^*(\Sigma)$ . D'après le lemme 3.7,  $\mathcal{V} = \mathcal{MT}^*(\Sigma_c)$ , qui est naturellement un langage temporisé régulier.
- (2) **Second cas.** Supposons que  $\mathcal{L} \neq \mathcal{MT}^*(\Sigma)$ . Afin d'obtenir une contradiction, supposons que  $\mathcal{V}$  est accepté par un automate temporisé  $\mathcal{A}$ . Soit  $y = (a_0, \tau_0) \dots (a_n, \tau_n) \in \mathcal{MT}^*(\Sigma) \setminus \mathcal{L}$ . Nous obtenons que pour tout mot temporisé  $w \in \mathcal{MT}^*(\Sigma)$ ,  $y.(c, \tau_n).(w + \tau_n) \in \mathcal{V}$  si et seulement si  $w \in \mathcal{R}_{pair}$ . Imitant la preuve de [BDGP98] du fait que  $\mathcal{R}_{pair}$  n'est pas régulier, nous allons obtenir la contradiction. Soit  $K$  la constante maximale apparaissant dans  $\mathcal{A}$ . Considérons le mot temporisé  $w' = y.(c, \tau_n).(a, \tau + \tau_n)$  où  $\tau \in \mathbb{N}$  est un entier naturel pair vérifiant  $\tau > K$ . Alors le mot  $w'$  est accepté par  $\mathcal{A}$ . En particulier la dernière transition d'une exécution acceptant  $w'$ , notée  $(\ell, g, a, R, \ell')$ , est telle que  $\ell' \in F$  est un état final. De plus, en notant  $(\ell, v)$  la configuration atteinte après que le préfixe  $y.(c, \tau_n)$  soit reconnu, alors la valuation  $v'$  atteinte juste avant le franchissement de la dernière transition vérifie  $v' = v + \tau$  et  $v' \models g$ . Grâce au choix de  $\tau$ , nous avons pour toute horloge  $x$  de  $\mathcal{A}$  l'inégalité  $v'(x) = v(x) + \tau > K$ . En particulier, pour tout entier naturel impair  $\tau'$  supérieur à  $\tau$ , le mot temporisé  $y.(c, \tau_n).(a, \tau_n + \tau')$  est également accepté par  $\mathcal{A}$ , ce qui fournit une contradiction. Ainsi,  $\mathcal{V}$  ne peut pas être accepté par un automate temporisé (sans transitions silencieuses).

Ceci conclut cette preuve :  $\mathcal{L}$  est universel sur  $\Sigma$  si et seulement si  $\mathcal{V}$  est un langage temporisé régulier.  $\square$

### 3.4 Déterminisation, complémentation

Dans [Fin06, Théorème 1], Finkel a prouvé que déterminer si le complément d'un langage temporisé régulier est régulier constitue un problème indécidable, de même que le problème de déterminer si un langage régulier peut être accepté par un automate temporisé déterministe. Nous étendons ces résultats à la classe des automates temporisés avec transitions silencieuses.

**Théorème 3.8** (Déterminisation). *Il est indécidable de déterminer si, étant donné un automate temporisé avec transitions silencieuses  $\mathcal{A}$ , il existe un automate temporisé déterministe (sans transitions silencieuses)  $\mathcal{B}$  tel que  $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})$ .*

Puisque les automates temporisés sans transitions silencieuses sont moins expressifs que les automates temporisés avec transitions silencieuses, le résultat précédent est une simple conséquence du résultat de Finkel.

**Théorème 3.9** (Complémentation).

1. *Il est indécidable de déterminer si, étant donné un automate temporisé avec transitions silencieuses  $\mathcal{A}$ , il existe un automate temporisé avec transitions silencieuses  $\mathcal{B}$  tel que  $\mathcal{L}(\mathcal{B}) = \overline{\mathcal{L}(\mathcal{A})}$ .*
2. *De plus ce résultat est vrai pour les automates temporisés définis sur des alphabets de deux lettres.*

La preuve de ce théorème n'est ni un corollaire de celui de Finkel (car la question posée est différente), ni une adaptation triviale de sa preuve. En effet sa preuve repose sur le fait qu'étant donné un mot temporisé et un automate temporisé, il existe un nombre fini d'exécutions de cet automate acceptant ce mot. Ceci n'est plus le cas pour les automates temporisés avec transitions silencieuses, comme cela a été mis en évidence dans la section 3.2. Nous proposons deux preuves de l'indécidabilité de ce problème. La première, plus simple, est correcte pour les automates définis sur des alphabets comportant au moins trois lettres et démontre ainsi le point (1) du théorème. La seconde,

plus technique, démontre le point (2) puisqu'elle est valable pour la classe des automates temporisés comportant exactement deux lettres.

Les deux preuves procèdent comme suit :

- Fixer un langage temporisé régulier  $\mathcal{L}$  ;
- Fixer un langage temporisé régulier  $\mathcal{R}$  dont le complémentaire  $\overline{\mathcal{R}}$  n'est pas régulier ;
- Construire à partir de  $\mathcal{L}$  et  $\mathcal{R}$  un nouveau langage temporisé régulier  $Compose(\mathcal{L}, \mathcal{R})$  (qui a été défini dans la section précédente) tel que  $\mathcal{L}$  est universel si et seulement si le complément de  $Compose(\mathcal{L}, \mathcal{R})$  est régulier.
- Réduire le problème de l'universalité à celui de la complémentation.

### 3.4.1 Cas d'un alphabet ayant trois lettres ou plus

Pour cette preuve, nous choisissons pour le langage  $\mathcal{R}$  le langage proposé dans [AM04] afin de démontrer aisément que la classe des langages temporisés réguliers n'est pas close par complémentation. En réalité, il apparaît que leur résultat, démontré dans le cadre des langages réguliers, est également valable pour les langages  $\varepsilon$ -réguliers, comme cela est établi dans la proposition suivante :

**Proposition 3.10.** *Supposons que  $\Sigma = \{a, b\}$ , et définissons  $\mathcal{R}_{a,b}$  comme le langage temporisé suivant :*

$$\mathcal{R}_{a,b} = \{w = (a_0, \tau_0) \dots (a_n, \tau_n) \in \mathcal{MT}^*(\Sigma) \mid \exists i, a_i = a, \text{ et } \forall j \geq i, \tau_j - \tau_i \neq 1\}.$$

*Ce langage est régulier, mais son complément n'est pas  $\varepsilon$ -régulier.*

La preuve de cette proposition est similaire à celle de [AM04], mais par souci de complétude, nous la détaillons.

**Preuve.** Le langage temporisé  $\mathcal{R}_{a,b}$  est accepté par l'automate temporisé représenté sur la figure 3.4, il est donc régulier.

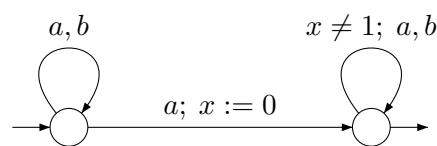


FIG. 3.4 – Un automate temporisé acceptant  $\mathcal{R}_{a,b}$

Nous montrons à présent que son complément n'est pas  $\varepsilon$ -régulier. Supposons au contraire qu'il existe un automate temporisé avec transitions silencieuses  $\mathcal{B}$  tel que  $\mathcal{L}(\mathcal{B}) = \overline{\mathcal{R}_{a,b}}$ . Le complément de  $\mathcal{R}_{a,b}$  est exactement l'ensemble des mots temporisés dans lesquels toute action  $a$  est suivie précisément une unité de temps plus tard d'une autre action ( $a$  ou  $b$ ).

Considérons  $\mathcal{T}_1$  l'ensemble des mots temporisés  $w$  sur  $\Sigma$  tels que :

- (i)  $\text{NON-TEMP}(w)$  appartienne au langage non temporisé régulier  $a^*b^*$ ,
- (ii) toutes les actions  $a$  interviennent dans l'intervalle  $[0, 1[$  et
- (iii) deux  $a$  ne peuvent pas intervenir à la même date.

Il est facile de vérifier que  $\mathcal{T}_1$  est un langage temporisé régulier. Observons à présent qu'un mot (non temporisé) de la forme  $a^n b^m$  appartient au langage  $\text{NON-TEMP}(\mathcal{T}_1 \cap \overline{\mathcal{R}_{a,b}})$  si et seulement si l'inégalité  $m \geq n$  est satisfaite. Ceci donne une contradiction. En effet nous savons d'une part que le langage  $\{a^n b^m \mid m \geq n\}$  n'est pas régulier, et d'autre part qu'il doit l'être car la régularité est préservée par intersection et par l'opérateur  $\text{NON-TEMP}$ . Le langage  $\overline{\mathcal{R}_{a,b}}$  ne peut donc pas être  $\varepsilon$ -régulier.  $\square$

Le lemme suivant sera utile pour la preuve du théorème 3.9.1. Il est d'une certaine façon un analogue de [AD94, Theorem 3.17] pour les compléments de langages temporisés.

**Lemme 3.11.** *Soit  $\mathcal{A}$  un automate temporisé (sans transitions silencieuses) sur un alphabet  $\Sigma$  et  $w \notin \mathcal{L}(\mathcal{A})$  un mot temporisé fini. Alors il existe un mot temporisé  $w' \notin \mathcal{L}(\mathcal{A})$  dont les dates sont rationnelles. De plus,  $\text{NON-TEMP}(w) = \text{NON-TEMP}(w')$ .*

**Preuve.** Soit  $d$  la granularité de  $\mathcal{A}$  et soit  $w = (a_1, \tau_1) \dots (a_n, \tau_n)$ . Pour simplifier les notations, nous définissons  $\tau_0 = 0$ . Nous construisons le mot temporisé  $w' = (a_1, \tau'_1) \dots (a_n, \tau'_n)$  par induction. De plus, ce mot temporisé  $w'$  devra satisfaire la contrainte suivante :

$$\forall 0 \leq i < j \leq n, \forall k \in \mathbb{N}, \tau_j - \tau_i \sim \frac{k}{d} \Leftrightarrow \tau'_j - \tau'_i \sim \frac{k}{d} \text{ avec } \sim \in \{<, \leq\} \quad (3.1)$$

La propriété d'induction est la suivante : il existe un mot temporisé  $w^m = (a_1, \tau_1^m) \dots (a_n, \tau_n^m)$  satisfaisant la propriété (3.1) avec  $\forall i \leq m, \tau_i^m \in \mathbb{Q}$ . Le cas de base est obtenu avec  $w^0 = w$ .

Supposons avoir construit le mot  $w^m = (a_1, \tau_1^m) \dots (a_n, \tau_n^m)$  vérifiant la propriété (3.1). Si  $\tau_{m+1}^m \in \mathbb{Q}$  alors  $w^{m+1} = w^m$  convient. Sinon, nous divisons l'ensemble d'indices  $I = \{0, 1, \dots, n\}$  en deux sous-ensemble  $I_{\equiv} = \{i \in I \mid \tau_i^m \equiv \tau_{m+1}^m \pmod{\frac{1}{d}}\}$  et  $I_{\neq} = I \setminus I_{\equiv}$ . Comme  $\tau_i^m \in \mathbb{Q}$  pour tout  $i \leq m$  (par hypothèse d'induction) et  $\tau_{m+1}^m \notin \mathbb{Q}$ , nous obtenons  $\{0, \dots, m\} \subseteq I_{\neq}$ . Soit  $\delta = \min(\min(\tau_i^m - \tau_{m+1}^m \pmod{\frac{1}{d}}, \tau_{m+1}^m - \tau_i^m \pmod{\frac{1}{d}} \mid i \in I_{\neq})$ . Remarquons que  $\delta > 0$ . Choisissons un certain  $\delta'$  tel que  $0 < \delta' \leq \delta$  et  $\tau_{m+1}^m + \delta' \in \mathbb{Q}$ . Nous construisons  $w^{m+1}$  comme suit.  $\forall i \in I_{\neq}, \tau_i^{m+1} = \tau_i^m$  et  $\forall i \in I_{\equiv}, \tau_i^{m+1} = \tau_i^m + \delta'$ . Il est facile de vérifier que le mot  $w^{m+1}$  ainsi défini satisfait la propriété d'induction.

Nous affirmons de plus que  $w' \notin \mathcal{L}(\mathcal{A})$ . Nous raisonnons par l'absurde ; supposons que  $w' \in \mathcal{L}(\mathcal{A})$  et fixons  $(\ell_0, v_0) \xrightarrow{\tau'_1} (\ell_0, v_0 + \tau'_1) \xrightarrow{e_1} (\ell_1, v_1) \dots (\ell_{n-1}, v_{n-1} + \tau'_n - \tau'_{n-1}) \xrightarrow{e_n} (\ell_n, v_n)$  une exécution finie acceptant le mot  $w'$ . Examinons l'exécution suivante :  $(\ell_0, v_0) \xrightarrow{\tau_1} (\ell_0, v_0 + \tau_1) \xrightarrow{e_1} (\ell_1, v_1) \dots (\ell_{n-1}, v_{n-1} + \tau_n - \tau_{n-1}) \xrightarrow{e_n} (\ell_n, v_n)$ . Considérons à présent une transition  $e_i$  de ces exécutions. La valeur de l'horloge  $x$  lors du franchissement de  $e_i$  dans la première exécution est  $\tau'_i - \tau'_j$  pour un certain indice  $j < i$  (correspondant à la dernière remise à zéro de  $x$  avant le franchissement de  $e_i$ ) et cette valeur dans la deuxième exécution est  $\tau_i - \tau_j$ . D'après la propriété (3.1) sur les relations entre les dates de  $w$  et celles de  $w'$ , l'observation précédente montre que la garde de toute transition  $e_i$  dans la deuxième exécution est vérifiée (car elle l'est dans la première exécution). Ceci implique que  $w \in \mathcal{L}(\mathcal{A})$ , ce qui est absurde.  $\square$

Nous pouvons à présent démontrer le théorème 3.9.1.

**Preuve du théorème 3.9.1.** Supposons que  $\{a, b\} \subseteq \Sigma$ . Nous considérons le langage  $\mathcal{R}_{a,b}$  introduit dans la proposition 3.10. Soit  $\mathcal{L} \subseteq \mathcal{MT}^*(\Sigma)$  un langage temporisé régulier. Définissons le langage temporisé  $\mathcal{V}$  sur  $\Sigma_c$  par  $\mathcal{V} = \text{Compose}(\mathcal{L}, \mathcal{R}_{a,b})$ . Nous affirmons que  $\mathcal{L} = \mathcal{MT}^*(\Sigma)$  si et seulement si  $\overline{\mathcal{V}}$  est accepté par un automate temporisé avec transitions silencieuses. Afin de démontrer cette affirmation, nous distinguons deux cas :

- (1) **Premier cas.**  $\mathcal{L}$  est universel sur  $\Sigma$ , i.e.  $\mathcal{L} = \mathcal{MT}^*(\Sigma)$ . D'après le lemme 3.7,  $\mathcal{V} = \mathcal{MT}^*(\Sigma_c)$ . Ainsi  $\overline{\mathcal{V}} = \emptyset$ , qui est naturellement  $(\varepsilon)$ -régulier.
- (2) **Second cas.**  $\mathcal{L}$  n'est pas universel, i.e.  $\mathcal{L} \neq \mathcal{MT}^*(\Sigma)$ . Afin d'obtenir une contradiction, supposons que  $\overline{\mathcal{V}}$  est accepté par un automate temporisé avec transitions silencieuses  $\mathcal{A}$  de granularité  $d$ . Fixons  $w = (a_0, \tau_0) \dots (a_k, \tau_k) \in \mathcal{MT}^*(\Sigma) \setminus \mathcal{L}$ . D'après le lemme 3.11, nous pouvons supposer que toutes les dates définissant  $w$  sont rationnelles. Nous définissons le langage temporisé régulier  $\mathcal{T}_2$  comme suit :

$$w' = w(c, \tau_k)x \in \mathcal{T}_2 \iff \begin{cases} \text{NON-TEMP}(x) \in a^*b^*, \\ x = (a, \tau'_0) \dots (a, \tau'_{k_1})(b, \tau''_0) \dots (b, \tau''_{k_2}), \\ \forall 0 \leq i \leq k_1, \tau'_i \in [\tau_k, \tau_k + 1[, \\ \forall 0 \leq i \neq j \leq k_1, \tau'_i \neq \tau'_j. \end{cases}$$

De même que pour la preuve de la proposition 3.10, observons que l'égalité suivante est valide :

$$\text{NON-TEMP}(\mathcal{T}_2 \cap \overline{\mathcal{V}}) = \{w' \mid \exists m \geq n, w' = \text{NON-TEMP}(w) c a^n b^m\}.$$

Ceci contredit l'hypothèse que  $\overline{\mathcal{V}}$  est  $\varepsilon$ -régulier puisque le membre droit de l'égalité précédente n'est pas régulier.

Ceci conclut donc la preuve :  $\mathcal{L}$  est universel si et seulement si  $\overline{\mathcal{V}}$  est  $\varepsilon$ -régulier.  $\square$

### 3.4.2 Cas d'un alphabet de deux lettres

Nous établissons dans un premier temps le lemme suivant :

**Lemme 3.12.** Soit  $\mathcal{A}$  un automate temporisé avec  $n$  horloges et de granularité  $d$ . Soit  $(\ell, v)$  une configuration de  $\mathcal{A}$ . Alors  $[0, \frac{1}{d}[$  peut être partitionné en  $I_1 \cup \dots \cup I_m$  où  $I_1, \dots, I_m$  sont des intervalles disjoints consécutifs tels que  $m \leq 2n + 1$ , et pour tout  $1 \leq j \leq m$ , pour tout  $\delta, \delta' \in I_j$ , pour tout  $k \in \mathbb{N}$ , pour tout  $x \in X$ , pour tout  $\bowtie \in \{<, \leq\}$ ,

$$v(x) + \delta \bowtie \frac{k}{d} \iff v(x) + \delta' \bowtie \frac{k}{d}.$$

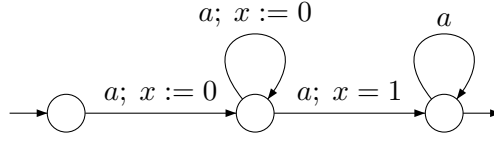
**Preuve.** Définissons l'ensemble  $\mathbb{Z}_d = \{\frac{k}{d} \mid k \in \mathbb{Z}\}$ . Pour tout  $x \in X$ , il existe exactement une valeur  $\delta_x \in [0, \frac{1}{d}[$  telle que  $v(x) + \delta_x \in \mathbb{Z}_d$ . Soit  $\Delta = \{\delta_1, \dots, \delta_J\}$  l'ensemble de ces valeurs ordonnées par ordre croissant ( $\delta_i < \delta_{i+1}$  pour tout  $i$ ). Celui-ci vérifie  $J \leq n$ . Considérons la partition de  $[0, \frac{1}{d}[$  définie par  $[0, \delta_1[ \uplus [\delta_1, \delta_1] \uplus \delta_1, \delta_2[ \uplus \dots \uplus \delta_J, \frac{1}{d}[$ . Il est facile de vérifier que cette partition satisfait les conditions énoncées dans le lemme.  $\square$

La proposition suivante étend aux automates temporisés avec transitions silencieuses le résultat standard énonçant que la classe des automates temporisés définis sur un alphabet à une lettre n'est pas close par complémentation.

**Proposition 3.13.** Soit  $\mathcal{R}_a$  le langage temporisé suivant :

$$\mathcal{R}_a = \{(a, \tau_1) \dots (a, \tau_n) \mid \exists 1 \leq i < j \leq n \text{ tel que } \tau_j - \tau_i = 1\}.$$

Le langage temporisé  $\mathcal{R}_a$  est régulier, mais son complémentaire  $\overline{\mathcal{R}_a}$  n'est pas  $\varepsilon$ -régulier.


 FIG. 3.5 – Un automate temporisé acceptant  $\mathcal{R}_a$ 

**Preuve.** Remarquons d'abord que le langage  $\mathcal{R}_a$  est accepté par l'automate temporisé représenté sur la figure 3.5. Afin d'obtenir une contradiction, supposons avoir trouvé un automate temporisé avec transitions silencieuses  $\mathcal{A}$  acceptant le langage temporisé  $\overline{\mathcal{R}_a}$ . Nous notons  $n$  son nombre d'horloges et  $d$  sa granularité. Le langage  $\overline{\mathcal{R}_a}$  est exactement l'ensemble des mots temporisés sur l'alphabet réduit au singleton  $\{a\}$  tels qu'aucune paire de  $a$  n'est séparée par exactement une unité de temps.

Choisissons un mot temporisé  $w = (a, \tau_1) \dots (a, \tau_{2N+1})$  dans  $\overline{\mathcal{R}_a}$  tel que  $N = 2n + 1$  et :

- pour tout  $1 \leq i < j \leq N$ ,  $0 < \tau_i < \tau_j < \frac{1}{d}$ ,
- pour tout  $1 \leq i \leq N$ ,  $1 < \tau_{N+i} < 1 + \tau_i < \tau_{N+i+1} < 1 + \frac{1}{d}$ .

Soit  $\varrho$  une exécution temporisée de  $\mathcal{A}$  acceptant  $w$  et considérons la configuration  $(\ell_{\varrho,1}^-, v_{\varrho,1}^-)$ . D'après le lemme 3.12 appliqué à cette configuration, nous obtenons une partition de  $[0, \frac{1}{d}[$  composée d'au plus  $N$  intervalles. Il existe donc un indice  $j$  appartenant à l'ensemble  $\{N + 1, \dots, 2N\}$  tel que  $\tau_j - 1$  et  $\tau_{j+1} - 1$  appartiennent au même intervalle de la partition.

Nous démontrons alors que pour toute horloge  $x \in X$  il existe  $k \in \mathbb{N}$  tel que :

$$\frac{k}{d} < v_{\varrho,\tau_j}^-(x) < v_{\varrho,\tau_j}^-(x) + (\tau_{j+1} - \tau_j) < \frac{k+1}{d} \quad (3.2)$$

Soit  $x \in X$ . Nous distinguons deux cas :

- ou bien  $x$  n'a pas été remise à zéro entre les configurations  $(\ell_{\varrho,1}^-, v_{\varrho,1}^-)$  et  $(\ell_{\varrho,\tau_j}^-, v_{\varrho,\tau_j}^-)$  le long de  $\varrho$ . Ceci implique que  $v_{\varrho,\tau_j}^-(x) = v_{\varrho,1}^-(x) + \tau_j - 1$ . Le choix de  $j$  entraîne que  $v_{\varrho,1}^-(x) + \tau_j - 1$  et  $v_{\varrho,\tau_j}^-(x) + (\tau_{j+1} - \tau_j) = v_{\varrho,1}^-(x) + \tau_{j+1} - 1$  vérifient les mêmes contraintes de « granularité  $d$  ». Ainsi l'équation (3.2) est vérifiée pour l'horloge  $x$ .
- ou bien au contraire l'horloge  $x$  a été remise à zéro le long de  $\varrho$  entre les deux configurations  $(\ell_{\varrho,1}^-, v_{\varrho,1}^-)$  et  $(\ell_{\varrho,\tau_j}^-, v_{\varrho,\tau_j}^-)$ . Dans ce cas, l'équation (3.2) est vérifiée pour  $k = 0$ . En effet,  $0 < v_{\varrho,\tau_j}^-(x)$  puisque  $\tau_j > 0$ . De plus, comme la date à laquelle l'horloge  $x$  a été remise à zéro pour la dernière fois entre les deux configurations mentionnées précédemment appartient à l'intervalle  $[1, 1 + \frac{1}{d}[$ , nous obtenons que  $v_{\varrho,\tau_j}^-(x) + (\tau_{j+1} - \tau_j) \leq (\tau_j - 1) + (\tau_{j+1} - \tau_j) < \frac{1}{d}$ , ce que nous souhaitons.

Définissons  $\delta = 1 + \tau_{j-N} - \tau_j$ . D'après l'équation (3.2) et les contraintes sur la séquence  $(\tau_i)_{1 \leq i \leq 2N+1}$  nous obtenons que pour toute horloge  $x \in X$ , il existe un  $k \in \mathbb{N}$  tel que :

$$\frac{k}{d} < v_{\varrho,\tau_j}^-(x) < v_{\varrho,\tau_j}^-(x) + \delta < \frac{k+1}{d} \quad (3.3)$$

Nous construisons à présent une exécution temporisée  $\varrho'$  comme suit. Elle coïncide avec  $\varrho$  jusqu'à la configuration  $(\ell_{\varrho,\tau_j}^-, v_{\varrho,\tau_j}^-)$ . Ensuite, elle laisse  $\delta$  unités de temps s'écouler, menant ainsi à la configuration  $(\ell_{\varrho,\tau_j}^-, v_{\varrho,\tau_j}^- + \delta)$ . Elle tire alors *instantanément* (*i.e.* en temps nul) la sous séquence de  $\varrho$  (disons  $\varrho_j$ ) menant de  $(\ell_{\varrho,\tau_j}^-, v_{\varrho,\tau_j}^-)$  à  $(\ell_{\varrho,\tau_j}^+, v_{\varrho,\tau_j}^+)$  (rappelons que  $\varrho$  est une séquence de  $\mathcal{A}$  qui peut donc contenir des transitions  $\varepsilon$ ). L'exécution temporisée  $\varrho_j$  n'est pas vide et contient au moins une transition étiquetée par  $a$ . Cette séquence peut également être exécutée à partir de  $(\ell_{\varrho,\tau_j}^-, v_{\varrho,\tau_j}^- + \delta)$

puisque d'après l'équation (3.3), les deux configurations appartiennent à la même région de l'automate des régions (défini dans la sous-section 1.4.3) de  $\mathcal{A}$ . Ainsi  $\varrho'$  peut exécuter les mêmes actions que  $\varrho$  avec éventuellement des délais différents (propriété de bisimulation à temps abstrait du graphe des régions) jusqu'à atteindre un état de contrôle acceptant (puisque  $\varrho$  est acceptante).

Par conséquent le mot temporisé accepté le long de  $\varrho'$  comporte deux occurrences de la lettre  $a$  séparées par une unité de temps (celles des dates  $\tau_{j-N}$  et  $1 + \tau_{j-N}$ ). Il n'appartient donc pas au langage  $\overline{\mathcal{R}_a}$ , ce qui fournit une contradiction.  $\square$

Nous pouvons maintenant démontrer le théorème 3.9.2 établissant l'indécidabilité dans le cadre des alphabets à deux lettres.

**Preuve du théorème 3.9.2.** La preuve suit les grandes lignes énoncées plus haut. Étant donné  $\mathcal{L}$  un langage temporisé régulier, définissons  $\mathcal{V} = \text{Compose}(\mathcal{L}, \mathcal{R}_a)$  où  $\mathcal{R}_a$  est le langage introduit dans la proposition 3.13. Nous affirmons que  $\mathcal{L} = \mathcal{MT}^*(\Sigma)$  si et seulement si  $\overline{\mathcal{V}}$  est accepté par un automate temporisé avec transitions silencieuses. Nous distinguons deux cas :

- (1) **Premier cas.** Supposons que  $\mathcal{L} = \mathcal{MT}^*(\Sigma)$ . D'après le lemme 3.7,  $\overline{\mathcal{V}} = \emptyset$  qui est bien ( $\varepsilon$ -)régulier.
- (2) **Second cas.** Supposons que  $\mathcal{L} \neq \mathcal{MT}^*(\Sigma)$ . Afin d'obtenir une contradiction, nous supposons que  $\overline{\mathcal{V}}$  est reconnu par un automate temporisé avec transitions silencieuses  $\mathcal{A}'$  de granularité  $d$  et ayant  $n$  horloges. Choisissons un mot  $w' = (a_1, \tau'_1) \dots (a_m, \tau'_m)$  dans  $\mathcal{MT}^*(\Sigma) \setminus \mathcal{L}$  et fixons un mot  $w = w'(c, \tau'_m)(a, \tau_1) \dots (a, \tau_{2N+1}) \in \mathcal{V}$  où  $N = 2n + 1$  qui vérifie :
  - pour tout  $1 \leq i < j \leq N$ ,  $\tau'_m < \tau_i < \tau_j < \tau'_m + \frac{1}{d}$  ;
  - pour tout  $1 \leq i \leq N$ ,  $\tau'_m + 1 < \tau_{N+i} < 1 + \tau_i < \tau_{N+i+1} < \tau'_m + 1 + \frac{1}{d}$ .

À partir d'une exécution temporisée  $\varrho$  acceptant  $w$  dans  $\mathcal{A}$ , nous construisons une nouvelle exécution temporisée  $\varrho'$ . Pour cela,  $\varrho'$  reconnaît le mot  $w$ , les  $N$  actions suivantes puis est obtenue en appliquant la construction de la preuve de la proposition 3.13 à partir de la configuration  $(\ell_{\varrho, \tau'_m+1}^-, v_{\varrho, \tau'_m+1}^-)$ . Il est alors facile de constater que le mot temporisé associé à  $\varrho'$  n'appartient pas à  $\mathcal{V}$ , produisant ainsi une contradiction.

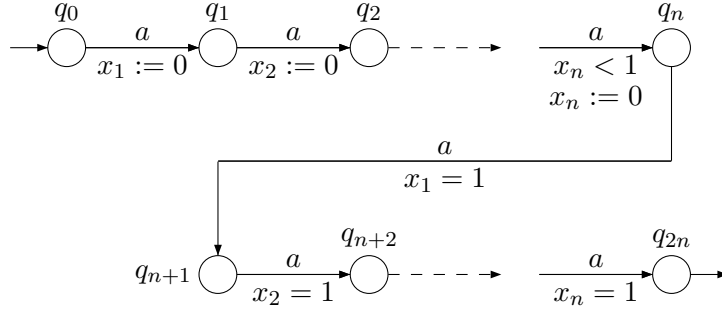
Ceci conclut la preuve du théorème 3.9.2.  $\square$

**Remarque 3.14** (Cas d'un alphabet à une lettre). Notons que le théorème 3.9 laisse le cas des alphabets à une lettre ouvert.  $\lrcorner$

### 3.5 Minimisation du nombre d'horloges

Dans [Fin06, Théorème 2], Finkel a démontré qu'étant donné un langage temporisé accepté par un automate temporisé ayant  $n$  horloges, déterminer si ce langage peut être accepté par un automate temporisé ayant  $n - 1$  horloges est indécidable. Dans cette section, nous démontrons que ce résultat est aussi valable dans le cadre des automates temporisés avec transitions silencieuses.

Nous démontrons dans un premier temps la proposition suivante qui exhibe une famille de langages temporisés telle que le  $n^{\text{e}}$  langage est accepté par un automate temporisé à  $n$  horloges, mais par aucun automate temporisé avec transitions silencieuses à  $n - 1$  horloges. Ces langages ont été introduits dans [HKWT95] pour obtenir un résultat similaire dans le cadre des automates temporisés. L'extension démontrée ici n'est pas triviale et requiert une analyse précise des exécutions temporisées.

FIG. 3.6 – Automate  $\mathcal{A}_n$  ayant  $n$  horloges.

**Proposition 3.15** (Langage ayant un nombre minimal d'horloges). *Soit  $n \geq 1$  un entier naturel strictement positif. Définissons le langage temporisé  $\mathcal{R}_n$  comme suit :*

$$\mathcal{R}_n = \{(a, \tau_1)(a, \tau_2) \dots (a, \tau_n) \mid \forall 1 \leq i \leq n, 0 \leq \tau_i < 1 \wedge \tau_{n+i} = 1 + \tau_i\}.$$

*Ce langage est accepté par un automate temporisé ayant  $n$  horloges, mais par aucun automate temporisé, même avec transitions silencieuses, ayant au plus  $n - 1$  horloges.*

**Preuve.** Soit  $n \geq 1$  un entier naturel strictement positif. Le langage  $\mathcal{R}_n$  est reconnu par l'automate temporisé  $\mathcal{A}_n$  représenté sur la figure 3.6.

Supposons à présent qu'il existe un automate temporisé avec transitions silencieuses  $\mathcal{B}$  ayant au plus  $n - 1$  horloges et vérifiant  $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})$ . Notons  $d$  sa granularité. Fixons des valeurs  $(\tau_i)_{1 \leq i \leq n}$  telles que  $0 < \tau_1 < \tau_2 < \dots < \tau_n < \frac{1}{d}$  et considérons le mot temporisé  $w = (a, \tau_1)(a, \tau_2) \dots (a, \tau_n)(a, \tau_1 + 1)(a, \tau_2 + 1) \dots (a, \tau_n + 1)$ . Clairement,  $w \in \mathcal{R}_n$  et donc  $w$  est accepté par  $\mathcal{B}$  le long d'une exécution  $\rho$ .

Pour tout indice  $i$  dans  $\{1, \dots, n\}$ , nous considérons la configuration  $(\ell_{\rho, \tau_i+1}^-, v_{\rho, \tau_i+1}^-)$ . Comme cela a été expliqué dans la section 3.2,  $\tau_i + 1 > 0$  implique que la dernière transition franchie avant d'atteindre cette configuration est une transition de délai. Nous distinguons deux cas :

- **Premier cas :** Il existe un indice  $i \in \{1, \dots, n\}$  pour lequel pour toute horloge  $x$ ,  $v_{\rho, \tau_i+1}^-(x) \not\equiv 0 \pmod{\frac{1}{d}}$ . Ceci implique que la région de  $\mathcal{B}$  (donc pour la granularité  $d$ ) à laquelle appartient la valuation  $v_{\rho, \tau_i+1}^-$  est « ouverte » du point de vue de l'écoulement du temps, *i.e.* que pour tout  $v \in r$ , il existe un certain  $\delta > 0$  tel que  $v + \delta \in r$  et  $v - \delta \in r$ . Ainsi, nous pouvons légèrement modifier le dernier écoulement de temps en ajoutant au délai une telle valeur  $\delta$ . La nouvelle configuration atteinte est la configuration  $(\ell_{\rho, \tau_i+1}^-, v_{\rho, \tau_i+1}^- + \delta)$  et de plus la valuation  $v_{\rho, \tau_i+1}^- + \delta$  appartient à la région contenant la valuation  $v_{\rho, \tau_i+1}^-$ . Appliquant la propriété de bisimulation à temps abstrait de l'automate des régions, nous obtenons que nous pouvons prolonger l'exécution depuis cette nouvelle valuation en suivant exactement les mêmes transitions discrètes que celles de  $\rho$  (éventuellement à des dates différentes). Ceci nous donne donc une nouvelle exécution acceptante. De plus, le mot temporisé accepté le long de cette exécution n'appartient pas à  $\mathcal{R}_n$  puisque le  $i^e$  et le  $i + N^e$   $a$  ne sont pas séparés par 1 unité de temps mais par  $1 + \delta$  unité de temps. Ceci fournit donc une contradiction.
- **Second cas :** Supposons au contraire que pour tout indice  $i \in \{1, \dots, n\}$ , il existe une horloge  $x$  telle que  $v_{\rho, \tau_i+1}^-(x) \equiv 0 \pmod{\frac{1}{d}}$ . Puisque le nombre d'horloges de  $\mathcal{B}$  est strictement inférieur à  $n$ , il existe une horloge  $x$  vérifiant  $v_{\rho, \tau_i+1}^-(x) \equiv 0 \pmod{\frac{1}{d}}$  et  $v_{\rho, \tau_j+1}^-(x) \equiv 0 \pmod{\frac{1}{d}}$  pour deux indices  $i$  et  $j$  tels que  $1 \leq i < j \leq n$ . Comme  $\tau_i + 1 > 0$  et  $\tau_j + 1 > 0$ , les deux valeurs  $v_{\rho, \tau_i+1}^-(x)$  et  $v_{\rho, \tau_j+1}^-(x)$  sont positives et donc égales à un certain  $\frac{k}{d}$  avec  $k \in \mathbb{N}^*$ . Ceci amène



à une contradiction puisque le temps écoulé entre les deux configurations correspondantes est strictement inférieur à  $\frac{1}{d}$  (et strictement positif). Le second cas ne peut donc pas se produire. Ceci conclut la preuve : un tel automate temporisé avec transitions silencieuses ne peut pas exister.  $\square$

Nous pouvons maintenant établir le théorème suivant, qui étend le théorème 2 de [Fin06] aux automates temporisés avec transitions silencieuses. Notons que nous obtenons l'indécidabilité pour les automates temporisés avec transitions silencieuses à *une* horloge alors que sans transitions silencieuses le problème est décidable pour les automates à une horloge et l'indécidabilité n'est obtenue qu'à partir de deux horloges [Fin06]. Ceci provient du fait que l'universalité est décidable pour les automates temporisés sans transitions silencieuses à une horloge tandis qu'elle est indécidable si les transitions silencieuses sont autorisées (voir rappels donnés dans le théorème 3.2).

**Théorème 3.16** (Minimiser le nombre d'horloges). *Soit  $n$  un entier naturel.*

- **Cas  $n \geq 2$ .** *Pour  $n \geq 2$ , il est indécidable de déterminer si, étant donné un automate temporisé à  $n$  horloges  $\mathcal{A}$  (et donc aussi si  $\mathcal{A}$  est un automate temporisé avec transitions silencieuses), il existe un automate temporisé avec transitions silencieuses  $\mathcal{B}$  ayant au plus  $n - 1$  horloges tel que  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ .*
- **Cas  $n = 1$ .** *Il est indécidable de déterminer si, étant donné un automate temporisé avec transitions silencieuses à une horloge  $\mathcal{A}$ , il existe un automate temporisé avec transitions silencieuses sans horloges  $\mathcal{B}$  tel que  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ .*

**Preuve.** Soit  $\Sigma$  un alphabet fini et soit  $a \in \Sigma$  une lettre. Soit  $n \geq 1$  un entier naturel strictement positif et  $\mathcal{R}_n$  le langage temporisé défini dans la proposition 3.15. Soit  $c$  une nouvelle lettre n'appartenant pas à  $\Sigma$ . Nous démontrons les deux parties du théorème simultanément (cas  $n = 1$  et  $n \geq 2$ ). Nous considérons un langage temporisé régulier (respectivement  $\varepsilon$ -régulier)  $\mathcal{L} \subseteq \mathcal{MT}^*(\Sigma)$  accepté par un automate temporisé ayant au plus  $n$  horloges si  $n \geq 2$  (respectivement par un automate temporisé avec transitions silencieuses ayant au plus une horloge si  $n = 1$ ). Nous construisons un autre langage temporisé  $\mathcal{V}_n$  sur  $\Sigma_c = \Sigma \cup \{c\}$  défini par  $\mathcal{V}_n = \text{Compose}(\mathcal{L}, \mathcal{R}_n)$ . D'après le lemme 3.7, ce langage est régulier (respectivement  $\varepsilon$ -régulier) et est accepté par un automate temporisé ayant  $n$  horloges (respectivement par un automate temporisé avec transitions silencieuses ayant une unique horloge). Nous affirmons que  $\mathcal{L}$  est universel si et seulement si  $\mathcal{V}_n$  est accepté par un automate temporisé avec transitions silencieuses ayant  $n - 1$  horloges (respectivement sans horloges). Nous distinguons deux cas :

1. **Premier cas.** Supposons  $\mathcal{L}$  universel sur  $\Sigma$ , *i.e.*  $\mathcal{L} = \mathcal{MT}^*(\Sigma^*)$ . Alors  $\mathcal{V}_n = \mathcal{MT}^*(\Sigma_c)$ , *i.e.*  $\mathcal{V}_n$  est universel sur  $\Sigma_c$  et peut donc être accepté par un automate temporisé sans horloges.
2. **Second cas.** Supposons que  $\mathcal{L}$  n'est pas universel sur  $\Sigma$ , *i.e.* que  $\mathcal{L}$  est strictement inclus dans  $\mathcal{MT}^*(\Sigma)$ . Alors il existe un mot temporisé  $u = (a_1, \tau_1) \dots (a_k, \tau_k) \in \mathcal{MT}^*(\Sigma)$  n'appartenant pas à  $\mathcal{L}$ . Considérons à présent un mot temporisé  $x \in \mathcal{MT}^*(\Sigma)$ . L'équivalence suivante est alors vérifiée :  $u.(c, \tau_k).(x + \tau_k) \in \mathcal{V}_n$  si et seulement si  $x \in \mathcal{R}_n$ . Afin d'obtenir une contradiction, supposons que  $\mathcal{V}_n$  soit accepté par un automate temporisé avec transitions silencieuses  $\mathcal{B}$  ayant  $n - 1$  horloges. Notons  $d$  sa granularité et fixons des valeurs  $(\tau'_i)_{1 \leq i \leq n}$  vérifiant  $0 < \tau'_1 < \tau'_2 < \dots < \tau'_n < \frac{1}{d}$ . nous considérons le mot temporisé  $v = (a, \tau'_1)(a, \tau'_2) \dots (a, \tau'_n)(a, \tau'_1 + 1)(a, \tau'_2 + 1) \dots (a, \tau'_n + 1)$ . Nous avons alors  $v \in \mathcal{R}_n$  et  $w = u.(c, \tau_k).(v + \tau_k) \in \mathcal{V}_n$  est donc accepté par  $\mathcal{B}$ . Nous pouvons alors appliquer le raisonnement développé dans la preuve de la proposition 3.15 au mot temporisé  $w$  et obtenir une contradiction. Cette preuve ne repose en effet pas sur le fait que dans la première configuration de la séquence, toutes les horloges ont une valeur nulle et elle peut donc être appliquée à partir de la configuration atteinte après

avoir reconnu le préfixe  $u.(c, \tau_k)$ . Nous pouvons donc conclure qu'un tel automate temporisé  $\mathcal{B}$  ne peut pas exister. Ainsi, le langage temporisé  $\mathcal{V}_n$  ne peut pas être reconnu par un automate temporisé avec transitions silencieuses ayant strictement moins de  $n$  horloges.

Nous avons donc démontré que déterminer si  $\mathcal{V}_n$  peut être reconnu par un automate temporisé avec transitions silencieuses ayant au plus  $n-1$  horloges est équivalent à décider si  $\mathcal{L}$  est universel. Comme les deux problèmes d'universalité correspondants (cas  $n \geq 2$  et  $n = 1$ ) sont indécidables (cf théorème 3.2), ceci conclut cette preuve.  $\square$

### 3.6 Opération de « mélange »

Dans cette section, nous nous intéressons à l'opération de « mélange ». Cette opération est standard dans le cas des mots non temporisés. Afin de la définir sur les mots temporisés, nous reprenons la représentation sous forme de mots de délais (définis page 28) [Dim05, Fin06]. En effet rappelons qu'un mot de délais est simplement un mot sur l'alphabet  $\mathbb{T} \times \Sigma$ .

Nous donnons dans un premier temps la définition usuelle de l'opération de mélange sur les mots (non temporisés) sur un alphabet  $X$  : étant donnés deux mots  $u, v \in X^*$  sur  $X$ , nous définissons le *mélange de  $u$  et  $v$* , noté  $u \sqcup v$ , par :

$$u \sqcup v = \{w = x_1 y_1 x_2 y_2 \dots x_n y_n \mid u = x_1 x_2 \dots x_n \text{ and } v = y_1 y_2 \dots y_n\}$$

où les  $x_i$  et les  $y_i$  peuvent être réduits à  $\varepsilon$ .

Cette définition est ensuite étendue aux ensemble de mots en définissant, pour  $S_1, S_2 \subseteq X^*$ , le mélange de  $S_1$  et  $S_2$  par :

$$S_1 \sqcup S_2 = \{s_1 \sqcup s_2 \mid s_1 \in S_1, s_2 \in S_2\}.$$

Nous pouvons alors appliquer ces définitions au cadre des mots de délais en utilisant l'alphabet  $X = \mathbb{T} \times \Sigma$ . Enfin, ces définitions s'appliquent également au cadre des mots temporisés en utilisant l'application Délai et son application réciproque  $\text{Délai}^{-1}$  : étant donnés  $u, v \in \mathcal{MT}^*(\Sigma)$ , nous définissons  $u \sqcup v$  par

$$u \sqcup v = \text{Délai}^{-1}(\text{Délai}(u) \sqcup \text{Délai}(v)).$$

Cependant, pour simplifier les démonstrations, nous présentons dans cette section tous les résultats dans le cadre des mots de délais. Nous définissons un « langage de délais » comme un sous-ensemble de  $(\mathbb{T} \times \Sigma)^\omega$ . Nous définissons le langage de délais accepté par un automate temporisé  $\mathcal{A}$  comme l'image par Délai de  $\mathcal{L}(\mathcal{A})$ . Nous obtenons naturellement les notions de langages de délais « réguliers » ou «  $\varepsilon$ -réguliers ».

Dima et Finkel ont démontré indépendamment que les langages de délais réguliers ne sont pas clos par opération de mélange [Dim05, Fin06]. Nous étendons d'abord ce résultat au cadre des langages de délais  $\varepsilon$ -réguliers.

**Proposition 3.17.** *Le mélange de deux langages de délais réguliers (et donc a fortiori  $\varepsilon$ -réguliers) n'est pas nécessairement  $\varepsilon$ -régulier.*

**Preuve.** Afin de démontrer ce résultat, nous suivons les lignes de la preuve de [Fin06]. Nous définissons d'abord trois langages de délais réguliers :

$$\begin{aligned} \mathcal{N}_1 &= \{(t_1, a).(1, a).(t_2, a) \mid t_1 + t_2 = 1\} \\ \mathcal{N}_2 &= \{(1, b).(s, b) \mid s \in \mathbb{T}\} \\ \mathcal{N}_3 &= \{(t_1, a).(1, b).(s, b).(1, a).(t_2, a) \mid t_1, s, t_2 \in \mathbb{T}\} \end{aligned}$$

Si le mélange de deux langages de délais réguliers était nécessairement  $\varepsilon$ -régulier et puisque les langages de délais  $\varepsilon$ -réguliers sont clos par intersection (car les langages temporisés réguliers le sont), alors le langage de délais  $(\mathcal{N}_1 \sqcup \mathcal{N}_2) \cap \mathcal{N}_3$  serait également  $\varepsilon$ -régulier. Nous montrons que ce n'est pas le cas.

$$(\mathcal{N}_1 \sqcup \mathcal{N}_2) \cap \mathcal{N}_3 = \{(t_1, a).(1, b).(s, b).(1, a).(t_2, a) \mid t_1, t_2, s \in \mathbb{T}, t_1 + t_2 = 1\}$$

Afin d'obtenir une contradiction, supposons qu'il existe un automate temporisé avec transitions silencieuses  $\mathcal{A}$  acceptant ce langage. Nous notons  $d$  sa granularité.

Soit  $w$  un mot de délais accepté par  $\mathcal{A}$  et tel que les propriétés suivantes sont vérifiées :

$$\begin{cases} t_2 & \not\equiv 0 \pmod{1/d} \\ s + t_2 & \not\equiv 0 \pmod{1/d} \\ s & \not\equiv 0 \pmod{1/d} \end{cases}$$

Puisque  $w$  est accepté par  $\mathcal{A}$  il existe une exécution dans cet automate qui accepte  $w$ , notons-la  $\varrho = \ell_0 \xrightarrow{e_1} \ell_1 \dots \ell_{n-1} \xrightarrow{e_n} \ell_n$  où  $e_i$  désigne une transition de  $\mathcal{A}$ . Cette exécution peut être vue comme un automate temporisé avec transitions silencieuses « linéaire »  $\mathcal{A}'$ . Décrivons comment définir  $\mathcal{A}'$ . Celui-ci contient  $n + 1$  états de contrôle correspondant aux occurrences des états de contrôle traversés par  $\varrho$ . Les horloges des deux automates sont identiques.  $\mathcal{A}'$  contient  $n$  transitions correspondant aux occurrences des transitions apparaissant le long de  $\varrho$ . La garde et la mise à jour d'une transition de  $\mathcal{A}'$  sont identiques à celles de la transition de  $\mathcal{A}$  correspondante. L'unique état final de  $\mathcal{A}'$  est l'état correspondant à  $\ell_n$ .

Par construction,  $\mathcal{A}'$  ne possède pas de cycle, vérifie  $w \in \mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$  et sa granularité  $d'$  divise celle de  $\mathcal{A}$ .

Appliquant [BDGP98, Théorème 21], il est possible de construire à partir de  $\mathcal{A}'$  un nouvel automate temporisé sans transitions silencieuses  $\mathcal{A}''$  acceptant le même langage temporisé et donc le même langage de délais. De plus la granularité de  $\mathcal{A}''$  est égale à celle de  $\mathcal{A}$ . Considérons à présent un chemin dans l'automate des régions de  $\mathcal{A}''$  acceptant le mot  $w$ . Grâce aux hypothèses faites sur  $s, t_1$  et  $t_2$ , la région atteinte juste avant le franchissement du dernier  $a$  est ouverte du point de vue de l'écoulement du temps. En effet il est facile de vérifier que toutes les horloges ont une valeur différente de 0 modulo la granularité de  $\mathcal{A}''$ . Une analyse élémentaire du mot temporisé montre que les valeurs possibles sont :  $t_2, t_2 + 1, t_2 + 1 + s, t_2 + 2 + s, 3 + s$  (rappelons qu'il n'y a plus de transitions silencieuses dans  $\mathcal{A}''$ ).

Par conséquent nous pouvons retarder le franchissement de ce troisième  $a$  d'une valeur strictement positive. Ceci suffit à obtenir un mot temporisé  $w'$  accepté par  $\mathcal{A}''$  mais n'appartenant pas à  $(\mathcal{N}_1 \sqcup \mathcal{N}_2) \cap \mathcal{N}_3$  (en effet il ne vérifie pas  $t_1 + t_2 = 1$ ). Ceci constitue donc une contradiction puisque  $w' \in \mathcal{L}(\mathcal{A}'') = \mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$ .  $\square$

Nous établissons à présent l'extension de [Fin06, Theorem 5] au cadre des automates temporisés avec transitions silencieuses. Dans la preuve de [Fin06], l'équivalence obtenue dans le second cas n'était pas correcte. Nous avons donc corrigé cette équivalence dans notre preuve en ajoutant une lettre  $b$ .

**Théorème 3.18** (Opération de mélange). *Déterminer si le mélange de deux langages de délais réguliers est  $\varepsilon$ -régulier est indécidable.*

**Preuve.** Soit  $\Sigma$  un alphabet fini contenant au moins la lettre  $a$ . Nous choisissons  $b$  et  $c$  deux lettres n'appartenant pas à  $\Sigma$  et notons  $\Sigma_b = \Sigma \cup \{b\}$  et  $\Sigma_c = \Sigma \cup \{c\}$ . Nous considérons un langage de

délais régulier  $\mathcal{L} \subseteq (\mathbb{T} \times \Sigma)^*$ . En utilisant le langage  $\mathcal{N}_1$  introduit dans la preuve de la proposition précédente, nous définissons le langage de délais  $\mathcal{V} \subseteq (\mathbb{T} \times \Sigma_c)^*$  comme l'union des trois langages de délais suivants : (ceci est une simple adaptation de *Compose* aux mots de délais)

$$\begin{aligned} \mathcal{V}_1 &= \{w \mid \exists w' \in \mathcal{L}, \exists w'' \in (\mathbb{T} \times \Sigma)^*, \exists \tau \text{ t.q. } w = w'.(c, \tau).w''\} \\ \mathcal{V}_2 &= \{w \mid |w|_c \neq 1\} \\ \mathcal{V}_3 &= \{w \mid \exists w' \in (\mathbb{T} \times \Sigma)^*, \exists w'' \in \mathcal{N}_1, \exists \tau \text{ t.q. } w = w'.(c, \tau).w''\} \end{aligned}$$

Puisque  $\mathcal{L}$  et  $\mathcal{N}_1$  sont réguliers,  $\mathcal{V}$  l'est également. Considérons à présent le langage  $\mathcal{W} = \mathcal{V} \sqcup \mathcal{N}_2$  où  $\mathcal{N}_2$  a été introduit dans la preuve précédente. Nous affirmons alors que  $\mathcal{L}$  est universel sur  $\Sigma$  si et seulement si  $\mathcal{W}$  est  $\varepsilon$ -régulier sur  $\Sigma \cup \{b, c\}$ . Nous distinguons deux cas :

1. **Premier cas.** Supposons que  $\mathcal{L}$  est universel sur  $\Sigma$ , i.e.  $\mathcal{L} = (\mathbb{T} \times \Sigma)^*$ . Alors  $\mathcal{V} = (\mathbb{T} \times \Sigma_c)^*$ , i.e.  $\mathcal{V}$  est universel sur  $\Sigma_c$ . Il est facile de vérifier que l'automate temporisé représenté sur la figure 3.7 accepte  $\mathcal{W}$ . En particulier,  $\mathcal{W}$  est donc ( $\varepsilon$ -)régulier.

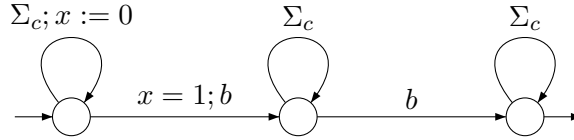


FIG. 3.7 – Un automate temporisé acceptant  $\mathcal{W}$ .

2. **Second cas.** Supposons que  $\mathcal{L}$  n'est pas universel sur  $\Sigma$ . Afin d'obtenir une contradiction, supposons que  $\mathcal{W}$  est  $\varepsilon$ -régulier. Alors le langage de délais  $\mathcal{X} = \mathcal{W} \cap ((\mathbb{T} \times \Sigma)^*. (1, c). \mathcal{N}_3)$  est  $\varepsilon$ -régulier. Choisissons un mot de délais  $w = (\tau_1, a_1) \dots (\tau_k, a_k) \in (\mathbb{T} \times \Sigma)^*$  qui n'appartient pas à  $\mathcal{L}$ . Considérons à présent un mot de délais  $x \in (\mathbb{T} \times \Sigma_b)^*$ . Nous allons montrer l'équivalence suivante :

$$w.(1, c).x \in \mathcal{X} \iff x \in (\mathcal{N}_1 \sqcup \mathcal{N}_2) \cap \mathcal{N}_3$$

Afin de prouver cette équivalence, supposons d'abord que  $w' = w.(1, c).x \in \mathcal{X}$ . Puisque  $w' \in (\mathbb{T} \times \Sigma)^*. (1, c). \mathcal{N}_3$ , nous obtenons  $x \in \mathcal{N}_3$ . D'autre part,  $w' \in \mathcal{W} = \mathcal{V} \sqcup \mathcal{N}_2$ . Puisqu'il y a une unique occurrence de  $c$  dans  $w'$ ,  $w'$  appartient ou bien au langage  $\mathcal{V}_1 \sqcup \mathcal{N}_2$  ou bien au langage  $\mathcal{V}_3 \sqcup \mathcal{N}_2$ . Nous allons montrer que seul le second cas peut survenir. Supposons que  $w' \in \mathcal{V}_1 \sqcup \mathcal{N}_2$ , alors  $w' \in w^-. (1, c). w^+ \sqcup w_2$  où  $w^- \in \mathcal{L}$  et  $w_2 \in \mathcal{N}_2$ . Ainsi  $w^- \neq w$  car  $w \notin \mathcal{L}$  et donc  $w$  est obtenu en insérant des occurrences des lettres de  $w_2$  dans  $w^-$ . Mais toutes ces occurrences ne peuvent être que des occurrences de  $b$  qui ne peuvent pas apparaître dans  $w$  car  $w$  est un mot sur  $\Sigma$ . En conclusion, nous avons  $w.(1, c).x \in ((\mathbb{T} \times \Sigma)^*. (1, c). \mathcal{N}_1) \sqcup \mathcal{N}_2$ . À nouveau, comme un mot de  $\mathcal{N}_2$  ne contient que des occurrences de la lettre  $b$ , nous obtenons  $x \in \mathcal{N}_1 \sqcup \mathcal{N}_2$ , ce qui conclut la preuve de la première direction. Réciproquement, la seconde implication découle facilement des définitions et de la remarque suivante :  $w.(1, c). (\mathcal{N}_1 \sqcup \mathcal{N}_2) \subseteq (w.(1, c). \mathcal{N}_1) \sqcup \mathcal{N}_2$ . Nous allons à présent adapter la preuve de la proposition 3.17 et montrer que  $\mathcal{X}$  n'est pas  $\varepsilon$ -régulier. Cependant nous ne pouvons pas appliquer cette preuve directement, et nous allons donc la mettre en œuvre à nouveau. Notons  $\mathcal{A}$  un automate temporisé avec transitions silencieuses acceptant  $\mathcal{X}$  et notons  $d$  sa granularité. Considérons un mot de

délais  $x$  appartenant à  $(\mathcal{N}_1 \sqcup \mathcal{N}_2) \cap \mathcal{N}_3$  tel que :

$$\begin{cases} t_2 & \not\equiv 0 \pmod{1/d} \\ s + t_2 & \not\equiv 0 \pmod{1/d} \\ s & \not\equiv 0 \pmod{1/d} \\ s + \sum_{j=i}^k \tau_j & \not\equiv 0 \pmod{1/d}, \forall i \in \{1, \dots, k\} \end{cases}$$

Ceci est possible car l'ensemble des paires  $(s, t_2)$  qui ne satisfont pas une de ces conditions est de mesure nulle dans le quart de plan  $\mathbb{T}^2$ . Nous pouvons donc ensuite considérer le mot de délais  $w' = w.(1, c).x \in \mathcal{X}$  qui est donc accepté par  $\mathcal{A}$ . Utilisant la même technique que dans la preuve précédente, nous construisons un nouvel automate temporisé sans transitions silencieuses  $\mathcal{A}''$  à partir de  $\mathcal{A}$  et  $w$  vérifiant :

- $\mathcal{A}''$  ne contient pas de transitions silencieuses,
- la granularité de  $\mathcal{A}''$  divise celle de  $\mathcal{A}$  et
- $w' \in \mathcal{L}(\mathcal{A}'') \subseteq \mathcal{L}(\mathcal{A})$ .

Nous explicitons le mot de délais  $w'$  :

$$w' = (\tau_1, a_1) \dots (\tau_k, a_k).(1, c).(t_1, a).(1, b).(s, b).(1, a).(t_2, a).$$

Une examination élémentaire de ce mot permet d'obtenir que les valeurs possibles pour les horloges de  $\mathcal{A}''$  lors du franchissement du dernier  $a$  sont les suivantes :  $t_2, t_2 + 1, t_2 + 1 + s, t_2 + 2 + s, 3 + s, 4 + s, 4 + s + \tau_k, \dots, 4 + s + \sum_{j=1}^k \tau_j$ . Par conséquent, les contraintes prises sur  $s, t_1, t_2$  et les  $\tau_i$  nous permettent d'obtenir que la région atteinte à cet instant est ouverte et qu'il est donc possible de retarder légèrement le franchissement de la dernière transition. Nous concluons comme précédemment : nous avons obtenu un nouveau mot  $w''$  accepté par  $\mathcal{A}''$  qui n'appartient pas à  $\mathcal{X}$ , puisqu'il viole la propriété  $t_1 + t_2 = 1$  imposée par  $(\mathcal{N}_1 \sqcup \mathcal{N}_2) \cap \mathcal{N}_3$ . Ceci fournit la contradiction escomptée.

Ceci conclut la preuve : déterminer si  $\mathcal{W}$  est  $\varepsilon$ -régulier est équivalent à décider si  $\mathcal{L}$  est universel.  $\square$

### 3.7 Extension aux mots infinis

Dans cette section, nous expliquons comment les résultats obtenus dans les sections précédentes peuvent s'étendre au cas de langages de mots infinis. En effet, les résultats précédents portaient sur les langages de mots finis. Les résultats que nous allons démontrer sont rassemblés dans un seul théorème.

**Théorème 3.19** (Mots infinis). *Les six problèmes suivants sont indécidables :*

1. *Étant donné un automate temporisé avec transitions silencieuses  $\mathcal{A}$ , déterminer s'il existe un automate temporisé  $\mathcal{B}$  tel que  $\mathcal{L}^\omega(\mathcal{B}) = \mathcal{L}^\omega(\mathcal{A})$ .*
2. *Étant donné un automate temporisé avec transitions silencieuses  $\mathcal{A}$ , déterminer s'il existe un automate temporisé déterministe  $\mathcal{B}$  tel que  $\mathcal{L}^\omega(\mathcal{B}) = \mathcal{L}^\omega(\mathcal{A})$ .*
3. *Étant donné un automate temporisé avec transitions silencieuses  $\mathcal{A}$  sur un alphabet d'au moins deux lettres, déterminer s'il existe un automate temporisé avec transitions silencieuses  $\mathcal{B}$  tel que  $\mathcal{L}^\omega(\mathcal{B}) = \overline{\mathcal{L}^\omega(\mathcal{A})}$ .*
4. *Étant donné un automate temporisé  $\mathcal{A}$  avec  $n$  horloges ( $n \geq 2$ ), déterminer s'il existe un automate temporisé avec transitions silencieuses  $\mathcal{B}$  avec  $n - 1$  horloges tel que  $\mathcal{L}^\omega(\mathcal{B}) = \mathcal{L}^\omega(\mathcal{A})$ .*

5. Étant donné un automate temporisé avec transitions silencieuses  $\mathcal{A}$  avec une unique horloge, déterminer s'il existe un automate temporisé avec transitions silencieuses  $\mathcal{B}$  sans horloges tel que  $\mathcal{L}^\omega(\mathcal{B}) = \mathcal{L}^\omega(\mathcal{A})$ .
6. Étant donnés deux automates temporisés  $\mathcal{A}$  et  $\mathcal{B}$ , déterminer si le mélange de  $\mathcal{L}^\omega(\mathcal{A})$  et  $\mathcal{L}^\omega(\mathcal{B})$  est  $\varepsilon$ -régulier<sup>1</sup>.

La preuve de ce théorème peut être obtenue à partir des preuves que nous avons proposées plus haut pour les théorèmes sur les mots finis. Le schéma pour obtenir chacun des points du théorème précédent est le même. L'idée consiste dans un premier temps à modifier la construction *Compose* afin de l'adapter au cadre des langages de mots infinis. Une fois cela réalisé, il s'agit de modifier les langages témoins ( $\mathcal{R}$ ) utilisés dans les preuves. Ainsi, il faut construire un langage de mots infinis témoignant de l'inclusion stricte entre les deux familles de langages temporisés auxquelles nous nous intéressons.

Comme précédemment, étant donné un alphabet  $\Sigma$ , nous choisissons une lettre  $c$  n'appartenant pas à  $\Sigma$  et notons  $\Sigma_c$  l'alphabet  $\Sigma \cup \{c\}$ .

**Définition 3.20.** Soit  $\mathcal{L} \subseteq \mathcal{MT}^*(\Sigma)$  et  $\mathcal{R} \subseteq \mathcal{MT}^\omega(\Sigma)$  deux langages temporisés sur  $\Sigma$  (le premier est un langage de mots finis tandis que le second est un langage de mots infinis). Alors le langage temporisé de mots infinis sur  $\Sigma_c$  *Inf-Compose*( $\mathcal{L}, \mathcal{R}$ ) est défini comme l'union des trois langages suivants :

$$\begin{aligned} \mathcal{V}_1 &= \{w \in \mathcal{MT}^\omega(\Sigma_c) \mid \exists w' \in \mathcal{L}, \exists w'' \in \mathcal{MT}^\omega(\Sigma), \exists \tau \text{ t.q. } w = w'(c, \tau)w''\} \\ \mathcal{V}_2 &= \{w \in \mathcal{MT}^\omega(\Sigma_c) \mid |w|_c \neq 1\} \\ \mathcal{V}_3 &= \{w \in \mathcal{MT}^\omega(\Sigma_c) \mid \exists w' \in \mathcal{MT}^*(\Sigma), \exists w'' \in \mathcal{R}, \exists \tau \text{ t.q. } w = w'(c, \tau)(w'' + \tau)\} \end{aligned}$$

Nous obtenons pour cette construction les propriétés suivantes, similaires à celles énoncées dans le lemme 3.7.

**Lemme 3.21.** Soit  $\mathcal{L} \subseteq \mathcal{MT}^*(\Sigma)$  et  $\mathcal{R} \subseteq \mathcal{MT}^\omega(\Sigma)$  deux langages temporisés sur un alphabet  $\Sigma$ .

- Si  $\mathcal{L}$  et  $\mathcal{R}$  sont acceptés par des automates temporisés (respectivement avec transitions silencieuses) ayant au plus  $n$  horloges, alors *Inf-Compose*( $\mathcal{L}, \mathcal{R}$ ) est également accepté par un automate temporisé (respectivement avec transitions silencieuses) ayant au plus  $n$  horloges.
- *Inf-Compose*( $\mathcal{MT}^*(\Sigma), \mathcal{R}$ ) =  $\mathcal{MT}^\omega(\Sigma_c)$ , il est donc accepté par un automate temporisé déterministe sans horloges.

La preuve de ce lemme est similaire à celle du lemme 3.7.

**Preuve du théorème 3.19.** Nous ne développons la preuve complète que pour le premier point, les autres points étant traités de façon similaire. Nous considérons une légère modification  $\mathcal{R}_{pair}^\omega$  du langage  $\mathcal{R}_{pair}$  définie par :

$$\mathcal{R}_{pair}^\omega = \{(a, \tau_1) \dots (a, \tau_n) \dots \mid \tau_i \equiv 0 \pmod{2} \text{ pour tout } i \geq 1\}.$$

Ce langage temporisé est accepté par l'automate temporisé avec transitions silencieuses représenté sur la figure 3.1 où l'ensemble des états répétés est réduit au singleton  $\{\ell\}$ .

Supposons que  $a \in \Sigma$  et fixons un langage temporisé régulier  $\mathcal{L} \subseteq \mathcal{MT}^*(\Sigma)$ . Considérons maintenant le langage temporisé  $\mathcal{V} = \text{Inf-Compose}(\mathcal{L}, \mathcal{R}_{pair}^\omega)$ . D'après le lemme 3.21, nous savons que  $\mathcal{V}$  est  $\varepsilon$ -régulier. Nous allons à présent démontrer que  $\mathcal{V}$  est régulier si et seulement si  $\mathcal{L}$  est universel sur  $\Sigma$ . Nous distinguons pour cela deux cas :

<sup>1</sup>Pour ce résultat, nous excluons les mots temporisés Zeno car la construction de [BDGP98] n'est valable que pour les mots infinis non Zeno.

- (1) **Premier cas.** Supposons  $\mathcal{L} = \mathcal{MT}^*(\Sigma)$ . D'après le lemme 3.21,  $\mathcal{V} = \mathcal{MT}^\omega(\Sigma_c)$  et  $\mathcal{V}$  est donc régulier.
- (2) **Second cas.** Supposons  $\mathcal{L} \neq \mathcal{MT}^*(\Sigma)$ . Afin d'obtenir une contradiction, supposons que  $\mathcal{V}$  soit accepté par un automate temporisé  $\mathcal{A}$ . Soit  $y = (a_0, \tau_0) \dots (a_n, \tau_n) \in \mathcal{MT}^*(\Sigma) \setminus \mathcal{L}$ . Nous avons alors pour tout mot temporisé  $w \in \mathcal{MT}^\omega(\Sigma)$ ,  $y.(c, \tau_n).(w + \tau_n) \in \mathcal{V}$  si et seulement si  $w \in \mathcal{R}_{pair}^\omega$ . Notons  $k$  la constante maximale apparaissant dans  $\mathcal{A}$  et considérons le mot temporisé  $w' = y.(c, \tau_n).(a, \tau + \tau_n).(a, \tau + \tau_n + 2) \dots$  où  $\tau \in \mathbb{N}$  est un entier naturel pair tel que  $\tau > K$ . Alors le mot temporisé  $w'$  est accepté par  $\mathcal{A}$  et il existe donc un chemin dans  $\mathcal{A}$  le long duquel  $w'$  est reconnu. Notons  $e = (\ell, g, a, U, \ell')$  la transition de ce chemin correspondant à la lettre  $a$  de la date  $\tau + \tau_n$  (élément  $(a, \tau + \tau_n)$  de  $w'$ ) et notons  $(\ell, v)$  la configuration atteinte juste après avoir reconnu  $y.(c, \tau_n)$ . La valuation lors du tir de  $e$  vaut alors  $v' = v + \tau$  et vérifie  $v' \models g$ . Grâce au choix de  $\tau$ , nous avons, pour toute horloge  $x$  de  $\mathcal{A}$ ,  $v'(x) = v(x) + \tau > K$ . En particulier pour tout entier naturel impair  $\tau'$  plus grand que  $\tau$ , le mot temporisé  $y.(c, \tau_n).(a, \tau_n + \tau')$  peut être accepté par le préfixe de ce chemin finissant par  $e$ . De plus, dans l'automate des régions, ce préfixe atteint la même région et peut donc être prolongé en un chemin acceptant pour un certain mot  $w''$  acceptant  $y.(c, \tau_n).(a, \tau_n + \tau')$  comme préfixe. Ceci est une contradiction puisque  $w''$  n'appartient pas à  $\mathcal{R}_{pair}^\omega$  car  $\tau'$  n'est pas pair. Finalement,  $\mathcal{V}$  ne peut pas être accepté par un automate temporisé.

Ceci conclut la preuve :  $\mathcal{L}$  est universel si et seulement si  $\mathcal{V}$  est un langage temporisé régulier.  $\square$

### 3.8 Conclusion

Dans ces travaux, nous avons étudié des problèmes de décision liés aux automates temporisés avec transitions silencieuses. D'abord, nous avons répondu négativement à une question centrale issue de l'introduction de transitions silencieuses : pouvons-nous décider si un langage accepté par un automate temporisé avec transitions silencieuses peut être accepté par un automate temporisé sans transitions silencieuses ? Ensuite, nous avons étendu des résultats d'indécidabilité connus jusqu'alors pour le cadre des automates temporisés sans transitions silencieuses. Les preuves de ces résultats sont plus délicates que les preuves existant pour le cadre des automates temporisés sans augmentent de façon significative les comportements possibles dans les automates temporisés. Ainsi, un même mot temporisé peut être accepté par un nombre fini de chemins s'il n'y a pas de transitions silencieuses et par une quantité non dénombrable de chemins sinon. Enfin, au-delà de l'intérêt de ces résultats, nous pensons que ces travaux permettent de mieux comprendre le rôle et l'impact des transitions silencieuses dans les automates temporisés.





## Chapitre 4

# Des automates temporisés étendus vers les réseaux de Petri temporels

Dans ce chapitre, nous présentons des travaux entrant dans le cadre de la comparaison entre les automates temporisés et les réseaux de Petri temporels. Plus précisément, le résultat central est une transformation efficace des automates temporisés étendus avec des mises à jour intégrales et des gardes diagonales vers les réseaux de Petri temporels. Cette transformation préserve les langages temporisés acceptés. L'essentiel des résultats présentés ici a été publié dans [BHR06a].

### 4.1 Introduction

Nous rappelons dans cette section le contexte dans lequel s'inscrit notre travail.

Afin d'appliquer des méthodes développées pour les réseaux de Petri temporels au cadre des automates temporisés et réciproquement, il est naturel de chercher à développer des transformations permettant de passer de l'un à l'autre de ces deux modèles. Selon les propriétés qui doivent être vérifiées par la suite sur ces modèles, les transformations doivent construire des modèles ayant certaines similitudes, *i.e.* dont les comportements des systèmes de transitions temporisés sous-jacents sont comparables. Il est donc naturel d'étudier le pouvoir expressif de chacun de ces modèles selon différents critères. Les critères auxquels nous nous intéressons ont été présentés dans la section 1.1 et sont la bisimilarité et l'équivalence de langages.

De nombreux travaux ont considéré ces problèmes et nous synthétisons ici les principaux résultats qui peuvent être trouvés dans la littérature, dans un ordre chronologique.

- Une première étude, publiée dans [BD99], a montré que la sous-classe des réseaux de Petri temporels « à la Merlin »<sup>1</sup> bornés est strictement moins expressive que les automates temporisés en termes de bisimilarité. Ce résultat, dont une preuve beaucoup plus simple est donnée dans [BCH<sup>+</sup>05b], repose sur le fait que dans un réseau de Petri temporel, une transition de délai ne peut pas rendre infranchissable une transition qui l'était. De plus, ce résultat est également valable pour la classe générale des réseaux de Petri temporels bornés.
- Un second travail, publié dans [HSLKT02], a montré l'équivalence en termes de langages des réseaux de Petri temporels saufs et des automates temporisés, en restreignant chacune de ces classes aux inégalités larges. La complexité de la transformation d'un automate temporisé en

---

<sup>1</sup>Rappelons que cette sous-classe correspond à la restriction dans laquelle tous les intervalles associés aux transitions doivent être fermés.

un réseau de Petri temporel sauf est quadratique. La transformation réciproque procède à partir du graphe d'accessibilité du réseau de Petri temporel sauf.

- Dans [CR03, CR06], une transformation structurelle (fondée uniquement sur la syntaxe) des réseaux de Petri temporels bornés en automates temporisés respectant la bisimilarité est présentée, démontrant ainsi que les automates temporisés sont au moins aussi expressifs que les réseaux de Petri temporels bornés pour la bisimulation. L'inclusion est stricte d'après le résultat rappelé précédemment.
- Une autre transformation est présentée dans [LR03, LR06]. Celle-ci construit également à partir d'un réseau de Petri temporel borné un automate temporisé qui lui est bisimilaire. Elle est fondée sur le graphe des classes.
- Plus récemment, [BCH<sup>+</sup>05c] présente une analyse des relations en termes d'expressivité entre les deux modèles et démontre en particulier les deux résultats suivants :
  1. les automates temporisés, les réseaux de Petri temporels saufs et les réseaux de Petri temporels bornés sont équivalents en termes de langages. De plus la transformation proposée pour la traduction d'un automate temporisé en un réseau de Petri temporel sauf est linéaire.
  2. les automates temporisés sont strictement plus expressifs que les réseaux de Petri temporels bornés en termes de bisimilarité,
- [BCH<sup>+</sup>05d] complète l'étude précédente en présentant une caractérisation (sémantique) de la sous-classe des automates temporisés bisimilaires aux réseaux de Petri temporels bornés.
- Avec un objectif semblable, [BPV06] propose une extension des réseaux de Petri temporels bornés à l'aide de priorités et démontre qu'une sous-classe de ce modèle est bisimilaire à une sous-classe importante des automates temporisés.
- Enfin, [DDSS07] propose une nouvelle construction d'un automate temporisé bisimilaire à un réseau de Petri temporel borné fondée cette fois sur le graphe des marquages du réseau de Petri sous-jacent. Cependant, cette méthode nécessite que le réseau de Petri sous-jacent soit également borné.

En termes de bisimilarité, la classe des automates temporisés est donc strictement plus expressive que celle des réseaux de Petri temporels bornés. Il n'est donc pas possible en toute généralité de construire, étant donné un automate temporisé, un réseau de Petri temporel borné qui lui est bisimilaire. En revanche, les deux classes sont équivalentes en termes de langages acceptés. Ceci justifie donc l'intérêt porté aux transformations d'automates temporisés en réseaux de Petri temporels équivalents du point de vue des langages et plus particulièrement aux transformations efficaces. Dans ces travaux, nous étendons le spectre d'application de cette étude en nous intéressant à la classe des automates temporisés étendus avec des mises à jour intégrales et des gardes diagonales. Afin d'alléger nos propos, nous dénommerons un élément de cette classe simplement par « automate temporisé étendu ».

Ce chapitre est organisé ainsi : dans la section 4.2, nous proposons une nouvelle transformation permettant de traduire de façon efficace un automate temporisé étendu par un réseau de Petri temporel sauf. Nous démontrons dans la sous-section 4.2.3 que cette transformation est quadratique dans le cas général, et linéaire si l'automate temporisé ne contient pas de gardes diagonales, ou pas de mises à jour intégrales. De plus, utilisant des résultats de concision obtenus pour ces automates temporisés étendus (Théorème 1.26), nous démontrons des résultats de concision similaires pour les réseaux de Petri temporels bornés. Dans la section 4.3, nous étendons la construction précédente au cadre des réseaux d'automates temporisés avec invariants. Enfin, nous discutons dans la section 4.4 les applications possibles de ces travaux.

## 4.2 Cas d'un automate temporisé étendu

Dans cette section, nous présentons une construction permettant d'obtenir, à partir d'un automate temporisé étendu, un réseau de Petri temporel équivalent du point de vue du langage accepté. Nous présenterons d'abord la transformation elle-même (sous-section 4.2.1), puis la preuve de correction de celle-ci (sous-section 4.2.2). Enfin, nous donnerons des résultats de complexité relatifs à cette construction, qui nous permettront de déduire des propriétés de concision (sous-section 4.2.3).

Nous supposons donné un automate temporisé étendu  $\mathcal{A} = (\Sigma, X, L, T, Inv, \ell_0, L_f, L_r)$ . Dans un premier temps, nous ne prenons pas en compte les invariants. En effet, nous avons vu dans la sous-section 1.4.2 que, du point de vue de l'accessibilité, il est possible de les retirer. Nous allons construire un réseau de Petri temporel équivalent  $\mathcal{N}$  d'une façon modulaire. Il est important de remarquer que ce réseau sera sauf par construction. De plus, les places portant le même nom seront partagées entre les différents modules de la construction. Les transitions ne portant pas d'étiquette correspondent à des transitions silencieuses (et sont donc implicitement étiquetées par  $\varepsilon$ ). Si l'intervalle de tir d'une transition est  $[0, 0]$ , la transition est représentée en noir et est dénommée *transition immédiate*. Son intervalle de tir n'est alors pas indiqué. De même, les intervalles  $[0, +\infty[$  ne seront pas indiqués. Une transition non coloriée en noir et ne portant pas d'intervalle sera donc étiquetée par  $[0, +\infty[$ . Enfin, une double flèche entre une place  $p$  et une transition  $t$  indique que  $p$  est à la fois en entrée et en sortie de la transition  $t$ .

### 4.2.1 Présentation de la construction

**Le module d'horloge.** Pour chaque horloge  $x$  de l'automate temporisé, nous construisons un sous-réseau qui enregistre et tient à jour la valeur de l'horloge  $x$ . Plus précisément, ce réseau mémorise à la fois la valeur de l'horloge (d'une façon implicite), et la valeur de vérité de toutes les contraintes  $x \sim c$  apparaissant dans l'automate. La valeur de vérité d'une telle contrainte est enregistrée explicitement, à l'aide d'une place  $T_{x \sim c}$ . Nous ajoutons également des places  $F_{x \sim c}$  si  $\sim$  est égal à  $<$  afin d'obtenir une place complémentaire. Pour toutes les mises à jour d'horloges  $y := h$  et les gardes diagonales  $x - y \sim c$  apparaissant dans l'automate, le réseau doit également prendre en compte la contrainte  $x \sim c + h$  et sa négation (sauf si celle-ci est équivalente à la contrainte tt ou ff – par exemple  $x > -3$  est équivalente à tt). En effet, la transformation que nous présentons va réaliser la construction de la partie 1.5.3 page 44, et celle-ci nécessite ces nouvelles contraintes. Enfin, ce module doit également prendre en compte les contraintes  $x \leq c$  et  $x \geq c$  si  $x := c$  est une mise à jour utilisée dans l'automate.

Le réseau représenté sur la figure 4.1 illustre notre construction dans le cas où  $x$  est comparée avec trois constantes  $\{c_1, c_2, c_3\}$  vérifiant  $c_1 < c_2 < c_3$ . Pour faciliter la lecture, nous supposons que 0 n'appartient pas à l'ensemble des constantes, mais ce cas peut être traité d'une façon similaire.

Expliquons à présent comment ce module simule l'écoulement du temps, comment il enregistre la valeur de l'horloge, et comment il mémorise la valeur de vérité des contraintes. Notons tout d'abord que toutes les places situées sur l'axe vertical (*i.e.* les places  $Avant_{c_1}^x, \acute{E}gal_{c_1}^x, \dots, Apr\grave{e}s_{c_3}^x$ ) sont mutuellement exclusives par construction. L'unique jeton qui est placé dans l'une de ces places, combiné avec l'âge<sup>2</sup> de la prochaine transition code la valeur de l'horloge  $x$ . Par exemple, si le jeton se trouve dans la place  $Avant_{c_2}^x$ , et si l'âge de la transition  $Atteint_{c_2}^x$  est  $\tau$ , alors la valeur de l'horloge  $x$  est  $c_1 + \tau$ . De même, l'horloge  $x$  vaudra  $c_2$  exactement dans les cas suivants :

- ou bien le jeton est dans la place  $Avant_{c_2}^x$ , et l'âge de  $Atteint_{c_2}^x$  est  $c_2 - c_1$ ,
- ou bien le jeton est dans la place  $\acute{E}gal_{c_2}^x$ ,

<sup>2</sup>Rappelons que l'âge d'une transition est la quantité de temps qui s'est écoulé depuis que cette transition est activée.

– ou bien enfin le jeton est dans la place  $Avant_{c_3}^x$  et l'âge de  $Atteint_{c_3}^x$  vaut 0.

Enfin, le réseau ne conserve pas la valeur exacte de  $x$  au-delà de la constante  $c_3$ , puisque c'est la constante maximale pour l'horloge  $x$ .

Les valeurs de vérité des contraintes sont mises à jour au fur et à mesure de l'évolution du réseau, tout en préservant les deux propriétés suivantes, qui sont fondamentales pour la correction de la construction :

1. Si la place  $T_{x \sim c}$  est marquée, alors la valeur correspondante de l'horloge  $x$ , disons  $v_x$ , satisfait

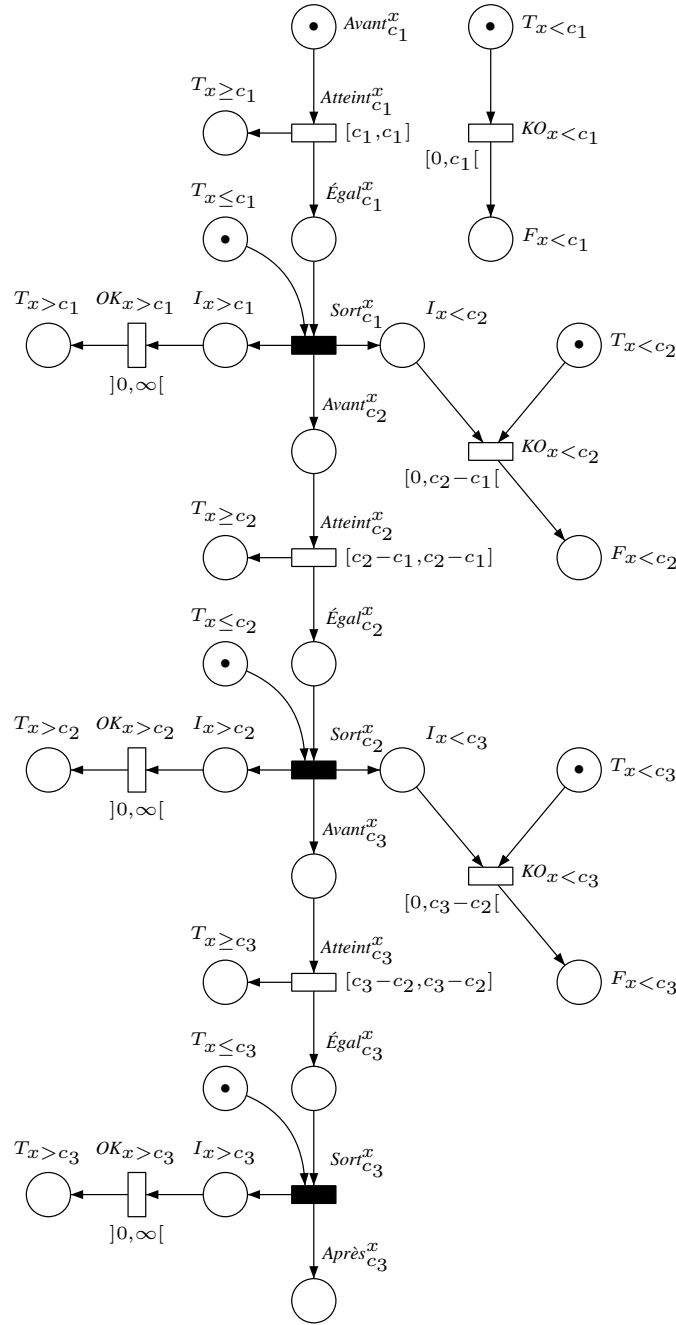


FIG. 4.1 – Le module d'évolution d'horloge (horloge  $x$ ).

la contrainte  $x \sim c$ , mais l'implication inverse n'est pas vraie,

2. Pour toute valeur possible  $v_x$  de  $x$ , il existe une exécution du réseau de durée  $v_x$  et telle que pour toute contrainte  $x \sim c$  vérifiée par  $v_x$ , la place  $T_{x \sim c}$  est marquée.

Enfin, remarquons que ce réseau ne tient pas compte des gardes diagonales, puisque celles-ci ne peuvent pas être traitées de la même façon (leur valeur de vérité reste inchangée lors de l'écoulement du temps).

Pour les résultats de complexité que nous développerons plus loin, il est important de noter que ce réseau a une taille linéaire dans le nombre de contraintes d'horloges impliquant  $x$  qu'il doit représenter (c.f. le début de sa description).

**Vider le module d'horloge.** Supposons qu'une transition de l'automate mette à jour l'horloge  $x$ . Le marquage du module associé à l'horloge  $x$  doit être mis à jour en conséquence, quelle que soit la

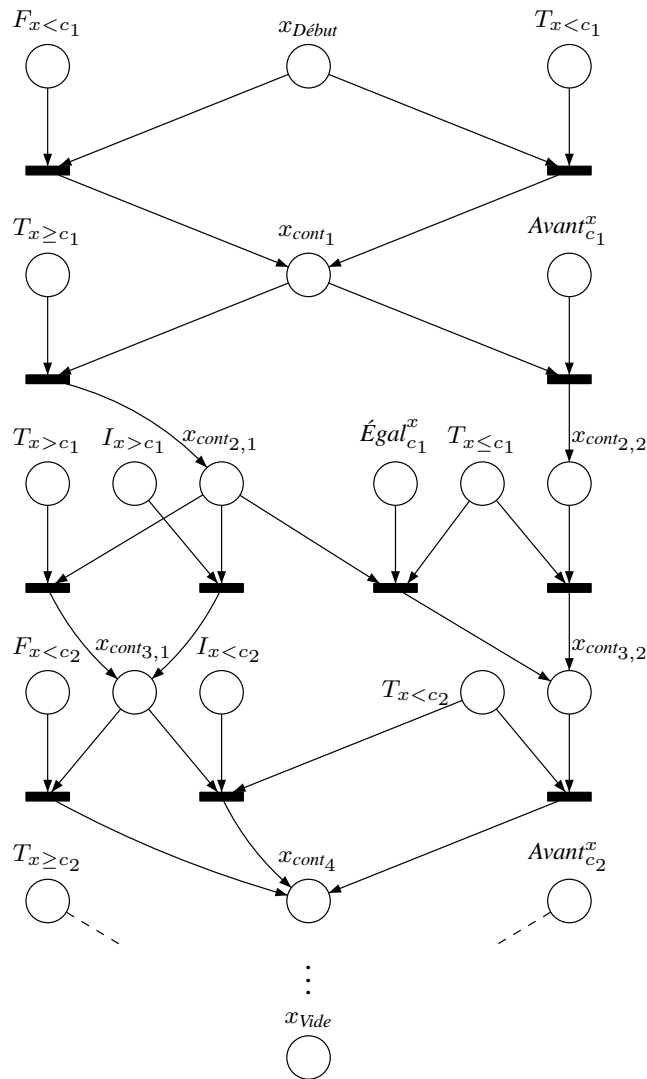


FIG. 4.2 – Vider le module d'horloge.

configuration courante.

Afin de coder une transition de l'automate temporisé, et de sorte à contrôler la taille du réseau de Petri temporel que l'on construit, nous allons procéder à la mise à jour du module d'horloge en deux étapes :

1. La première étape est représentée sur la figure 4.2 et consiste à retirer tous les jetons présents dans le module d'horloge en les consommant ;
2. La seconde étape est discutée dans le paragraphe suivant, et consiste à marquer les places appropriées dans le module d'horloge.

Afin de retirer tous les jetons présents dans le réseau représenté sur la figure 4.1, nous allons les consommer de haut en bas. Les places de contrôle du réseau de la figure 4.2 (précisément  $\{x_{Début}, x_{cont_1}, x_{cont_{2,1}}, x_{cont_{2,2}}, \dots, x_{Vide}\}$ ) ordonnent le processus de consommation, et mémorisent certaines informations de sorte à éviter un saut quadratique de complexité lié au nombre de transitions.

Décrivons en partie le comportement du réseau de la figure 4.2. En premier lieu, il retire le jeton qui se trouve ou bien dans la place  $F_{x < c_1}$  ou bien dans la place  $T_{x < c_1}$  (ces deux places sont mutuellement exclusives, à cause de la transition  $KO_{x < c_1}$ ). Ensuite, il retire le jeton qui se trouve dans une des deux places  $Avant_{c_1}^x$  et  $T_{x \geq c_1}$  (ces deux places sont également mutuellement exclusives à cause de la transition  $Atteint_{c_1}^x$ ). Grâce aux places de contrôle du réseau (les places  $x_{cont_{2,1}}$  et  $x_{cont_{2,2}}$ ), nous savons quel cas s'est produit, *i.e.* dans quelle place se trouvait le jeton. S'il se trouvait dans la place  $Avant_{c_1}^x$ , il n'y aura pas de jeton dans les places  $T_{x > c_1}$  et  $I_{x > c_1}$ . Au contraire, s'il se trouvait  $T_{x \geq c_1}$ , alors il y aura ou bien deux jetons dans les places  $Égal_{c_1}^x$  et  $T_{x \leq c_1}$ , ou bien un dans la place  $I_{x > c_1}$ , ou bien enfin un dans la place  $T_{x > c_1}$ . Le reste du module distingue donc ces différents cas.

Ces remarques nous permettent de borner la largeur du module qui « vide » le module d'horloge. Plus précisément, à un certain « niveau », il y a au plus 4 places qui sont concurrentes. Ceci permet ainsi de borner la taille de ce deuxième module. Enfin, remarquons que ce module est déclenché par l'arrivée d'un jeton dans la place  $x_{Début}$  et que le module d'horloge est « vidé » lorsqu'un jeton arrive dans la place  $x_{Vide}$ .

Pour les résultats de complexité, il reste à noter que la taille de ce module est linéaire dans la taille du réseau précédent.

**Mise à jour du marquage du module d'horloge.** Nous souhaitons mettre à jour le marquage des places codant les valeurs de vérité des contraintes d'horloges associées au module d'horloge quand cette horloge est mise à jour à une certaine valeur  $c$ . Une méthode naïve consisterait à décrire, pour chaque mise à jour possible, un module permettant de mettre à jour le module d'horloge. Cependant, nous voulons contrôler la taille du réseau de Petri temporel résultant, et nous allons pour cela construire un seul réseau capable d'effectuer toutes les mises à jour pour  $x$ , le nouveau marquage dépendant bien sûr de la valeur  $c$  affectée à  $x$ .

L'idée de notre construction est la suivante : quand l'horloge  $x$  est affectée à  $c$ , alors la contrainte  $x \leq c$  est satisfaite, et, en conséquence, toutes les contraintes supérieures plus grandes sont également satisfaites ( $x \prec c'$ , pour  $\prec \in \{<, \leq\}$  et  $c' > c$ ). Nous allons donc construire une chaîne de propagation pour les contraintes supérieures qui respecte les implications précédentes. Naturellement, le même raisonnement s'applique pour les contraintes inférieures, et une autre chaîne est utilisée pour celles-ci.

Les deux réseaux de propagation sont représentés sur la figure 4.3, et tirent profit des remarques précédentes. Les deux chaînes causales sont représentées par deux composantes connexes distinctes. Afin de déclencher ce réseau lors de la mise à jour de l'horloge  $x$  à la valeur  $c_i$  (*i.e.* lors d'une opération  $x := c_i$ ), deux jetons sont produits dans les places  $Inf_{x:=c_i}$  et  $Sup_{x:=c_i}$ . Pour le réseau de

gauche (respectivement de droite), la mise à jour du marquage se termine lorsqu'un jeton atteint la place  $Inf_{x:=0}$  (respectivement la place  $Sup_{x_{fin}}$ ).

Jusqu'à présent, nous n'avons pas encore placé le jeton dans une des places de l'axe vertical du module d'horloge, qui code implicitement la valeur de l'horloge. Ceci sera réalisé par le réseau simulant le tir d'une transition de l'automate temporel (*c.f.* le dernier paragraphe de cette description).

Enfin, concernant la complexité de la transformation, la taille de ce réseau est à nouveau linéaire dans la taille du module d'horloge.

**Contraintes d'horloges diagonales.** La valeur de vérité d'une contrainte diagonale  $x - y \sim h$  (qui est invariante par écoulement du temps) est représentée par deux places mutuellement exclusives  $T_{x-y \sim h}$  et  $F_{x-y \sim h}$ . Nous construisons un sous-réseau pour chaque contrainte atomique  $x - y \sim h$  et chaque mise à jour de l'une ou l'autre des horloges  $x$  et  $y$ .

La figure 4.4 représente le réseau correspondant à la contrainte diagonale  $x - y \leq h$  et la mise à jour  $y := h'$ . Lorsque  $y$  est affectée à  $h'$ , la valeur de vérité de  $x - y \leq h$  doit être mise à jour en tenant compte de la valeur de vérité de la contrainte (non diagonale)  $x \leq h + h'$ .

Les places  $\{Diag_i^{y:=h'}\}_{i=1..d(y)+1}$  contrôlent la mise à jour des sous-réseaux correspondant à des

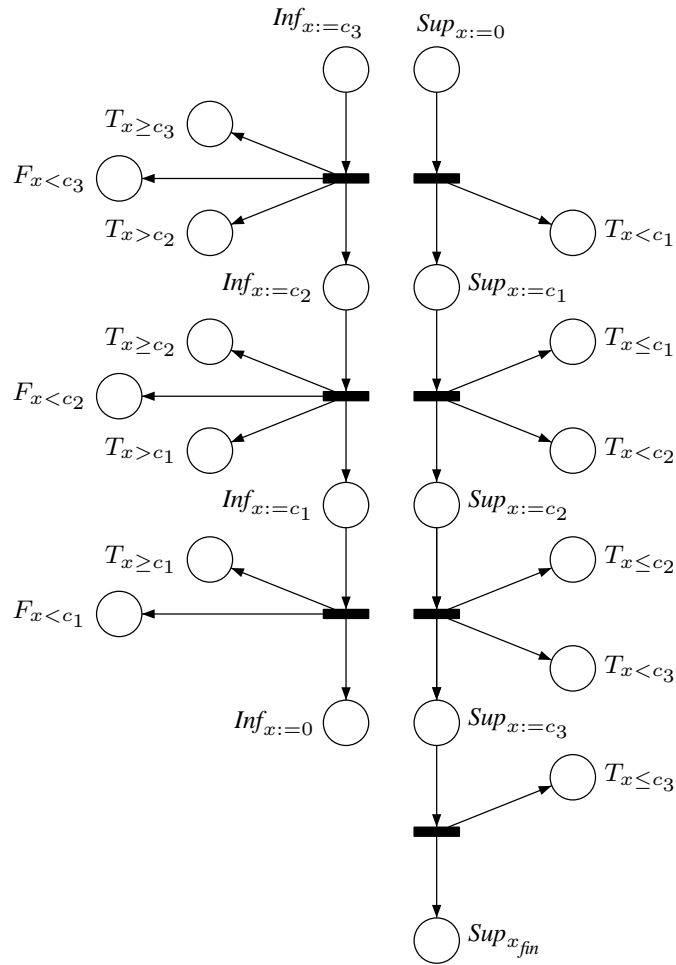


FIG. 4.3 – Mise à jour du marquage du module d'horloge.

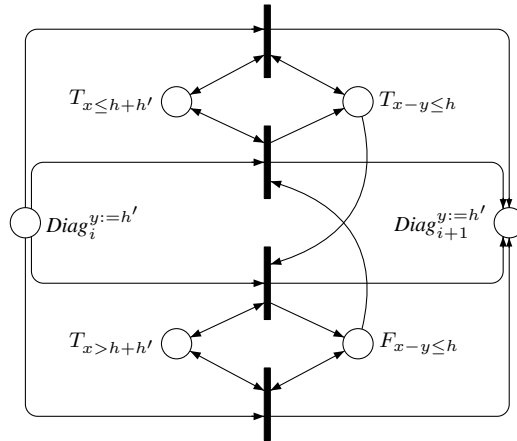


FIG. 4.4 – Le sous-réseau pour  $x - y \leq h$  et  $y := h'$

gardes diagonales faisant intervenir l'horloge  $y$  ( $d(y)$  dénote le nombre de telles contraintes). Dans la figure 4.4,  $i$  est l'indice de la contrainte  $x - y \leq h$  ( $1 \leq i \leq d(y)$ ).

Enfin, il est important de noter que pour chaque garde diagonale  $x - y \leq h$  et chaque mise à jour  $y := h'$ , la taille du sous-réseau associé qui est construit est constante (c.f. figure 4.4). Par ailleurs, le nombre de tels sous-réseaux est proportionnel au nombre de combinaisons d'une garde diagonale et d'une mise à jour, i.e. dans le pire des cas, quadratique. De plus, si nous ne considérons que l'une ou l'autre des deux extensions, alors ce nombre devient linéaire. En effet, si les seules mises à jour autorisées sont celles de la forme  $y := 0$ , alors le nombre de tels modules est linéaire dans le nombre de contraintes diagonales. Enfin, si les contraintes diagonales ne sont pas autorisées, alors il n'est pas nécessaire de construire ces modules.

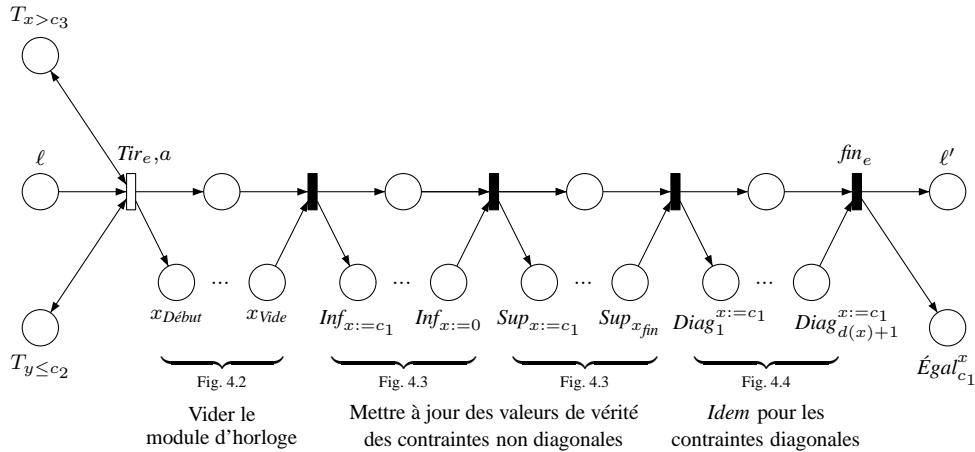


FIG. 4.5 – Simulation de la transition  $e = (\ell, x > c_3 \wedge y \leq c_2, a, x := c_1, \ell')$

**Simulation des transitions de l'automate temporisé.** Nous associons à chaque état de contrôle  $\ell \in L$  de l'automate une place éponyme dans le réseau de Petri temporel. La place  $\ell$  est initialement marquée si et seulement si  $\ell$  est un état initial de l'automate. Afin de simuler le tir d'une transition  $e = (\ell, g, a, \mu, \ell')$ , nous devons d'abord vérifier que la transition est bien franchissable, i.e. que la



contrainte d'horloges  $g$  est vérifiée. Pour cela, nous nous intéressons aux contraintes atomiques qui constituent  $g$ , à savoir les contraintes  $(g_i)_{1 \leq i \leq m(e)}$ , en écrivant  $g = g_1 \wedge \dots \wedge g_{m(e)}$ . Pour tester si ces contraintes atomiques sont satisfaites, nous utilisons les places  $T_{g_i}$  des modules d'horloges correspondants (ou des modules de gardes diagonales s'il s'agit de contraintes diagonales). Ensuite, nous mettons à jour successivement chacun des réseaux, en prenant en compte les opérations de mises à jour indiquées. Nous considérons donc donnée une mise à jour donnée par un élément  $\mu \in (\mathbb{N} \cup \{\perp\})^X$  où  $X$  dénote l'ensemble des horloges de l'automate temporisé  $\mathcal{A}$ . Rappelons que la sémantique de cette mise à jour est qu'elle affecte la valeur  $\mu(x)$  à l'horloge  $x$  si  $\mu(x) \neq \perp$  et laisse  $x$  inchangée sinon. La prise en compte de cette mise à jour est réalisée à l'aide du sous-réseau représenté sur la figure 4.5 pour une transition  $e = (\ell, x > c_3 \wedge y \leq c_2, a, \mu, \ell')$  où, pour simplifier la représentation du module, nous supposons que  $\mu$  est définie par  $\mu(x_1) = c_1$  et  $\mu(x) = \perp$  pour  $x \neq x_1$ . La transition  $Tir_e$  est étiquetée par l'action  $a$  (ce qui est représenté par la notation  $Tir_e, a$ ). Il est également important de remarquer que la place correspondant à la position de l'horloge ( $\acute{E}gal_{c_1}^x$ ) est marquée à l'issue de l'exécution de ce sous-réseau.

Ce sous-réseau est de taille linéaire dans la taille de la transition de l'automate temporisé d'origine.

**Places finales et répétées.** Les places finales et répétées du réseau de Petri temporel construit coïncident simplement avec les ensembles  $L_f$  et  $L_r$  d'états de contrôle finals et répétés de l'automate temporisé  $\mathcal{A}$ .

Nous expliquons à présent en quoi notre construction diffère de celle proposée dans [BCH<sup>+</sup>05c]. La prise en compte de l'écoulement du temps ainsi que l'évolution des horloges sont traitées différemment : au lieu d'avoir un réseau de petite taille par contrainte d'horloge apparaissant dans l'automate, nous avons seulement un réseau par horloge de l'automate qui permet de retenir la valeur de l'horloge, ainsi que les valeurs de vérité de toutes les contraintes liées à cette horloge. Notre méthode requiert une construction plus sophistiquée lors de la mise à jour des valeurs de vérité des contraintes afin d'éviter une explosion de la taille du réseau de Petri temporel construit. En effet, un traitement naïf de ces mises à jour donnerait des sous-réseaux de taille quadratiques, mais la technique développée ici qui consiste à vider le réseau, puis à replacer des jetons aux places nécessaires d'une façon descendante permet de conserver des sous-réseaux de taille linéaire. Grâce à cela, notre technique permet de traiter le cas des contraintes diagonales et des mises à jour intégrales.

### 4.2.2 Preuve de correction

La preuve de correction repose sur l'existence de deux simulations, une impliquant l'inclusion du langage accepté par l'automate temporisé dans le langage accepté par le réseau de Petri temporel, et l'autre impliquant l'inclusion inverse. Soit  $\mathcal{A}$  un automate temporisé étendu avec des mises à jour intégrales et gardes diagonales, et soit  $\mathcal{N}$  le réseau de Petri temporel obtenu après l'application de la construction décrite dans la sous-section précédente.

**Preuve de  $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{N})$ .** Nous définissons une relation  $\mathcal{R}$  entre les configurations de  $\mathcal{A}$  et celles de  $\mathcal{N}$  définie ainsi :  $(\ell, v) \mathcal{R} (M, \nu)$  si et seulement si les conditions suivantes sont satisfaites. Soit  $x$  une horloge, et notons  $\mathcal{C}_{\mathcal{N}}(x) = \{c_1, \dots, c_n\}$  l'ensemble des constantes apparaissant dans  $\mathcal{A}$  et liées à  $x$ , triées par ordre croissant. Définissons  $c(x) = \inf\{c_j \mid c_j \geq v(x)\}$  avec la convention que  $c(x) = \infty$  si cet ensemble est vide. Les conditions suivantes doivent alors être vérifiées :

- Si  $c(x) = v(x)$  alors  $M(\acute{E}gal_{c(x)}^x) = 1$  ;
- Sinon, si  $v(x) < c(x) < \infty$ , alors  $M(\text{Avant}_{c(x)}^x) = M(I_{x < c(x)}) = 1$ , et  $\nu(\text{Atteint}_{c(x)}^x) = \nu(KO_{x < c(x)}) = c(x) - v(x)$  ;
- Sinon,  $M(\text{Après}_{c_n}^x) = 1$  (cas  $c(x) = \infty$ ).

Nous imposons de plus les contraintes suivantes :

- pour toute place  $T_{x \sim c}$  telle que  $v(x) \sim c$ ,  $M(T_{x \sim c}) = 1$ ,
- pour toute place  $F_{x < c}$  telle que  $\neg(v(x) < c)$ ,  $M(F_{x < c}) = 1$ ,
- pour toute place  $T_{x-y \sim c}$  telle que  $v(x) - v(y) \sim c$ ,  $M(T_{x-y \sim c}) = 1$ ,
- pour toute place  $F_{x-y \sim c}$  telle que  $\neg(v(x) - v(y) \sim c)$ ,  $M(F_{x-y \sim c}) = 1$ ,
- et enfin,  $M(\ell) = 1$ .

Le marquage des autres places doit être nul, et la configuration  $(M, \nu)$  doit être admissible, *i.e* que les âges des transitions activées non décrites plus haut doivent être dans les intervalles des valeurs admissibles.

Nous observons tout d'abord que  $(\ell_0, \mathbf{0}) \mathcal{R}(M_0, \nu_0)$  et supposons que  $(\ell, \nu) \mathcal{R}(M, \nu)$ .

*Premier cas.* Simulation d'une transition de délai  $(\ell, \nu) \xrightarrow{d} (\ell, \nu + d)$ . Soit  $X'$  le sous-ensemble formé des horloges  $x$  telles que  $v(x) \in \mathcal{C}_{\mathcal{N}}(x)$ . Soit de plus  $X''$  le sous-ensemble formé des horloges  $x \notin X'$  telles que  $\inf\{c - v(x) \mid c \in \mathcal{C}_{\mathcal{N}}(x) \wedge c > v(x)\}$  est minimal. Nous notons  $\tau$  cette valeur (remarquons que  $\tau = \infty$  si  $X'' = \emptyset$ ) et enfin  $c(x)$  la constante associée à l'horloge  $x$ , pour chaque horloge  $x \in X''$ .

Nous décomposons les transitions de délai pour n'avoir que les cas suivants à considérer :

- $X' = \emptyset$  et  $d < \tau$ . Alors  $(M, \nu) \xrightarrow{d} (M, \nu + d)$  et  $(\ell, \nu + d) \mathcal{R}(M, \nu + d)$
- $X' = \emptyset$  et  $d = \tau$ . Dans ce cas, pour tout  $x \in X''$ , nous tirons la transition  $KO_{x < c(x)}$  puis nous laissons une durée  $d$  s'écouler. D'autre part, pour tout  $x \in X''$ , nous tirons la transition  $Atteint_{c(x)}^x$ . La configuration atteinte  $(M', \nu')$  vérifie alors  $(\ell, \nu + d) \mathcal{R}(M', \nu')$ .
- $X' \neq \emptyset$  et  $d < \tau$ . Dans ce cas, pour tout  $x \in X'$ , nous tirons la transition  $Sort_{v(x)}^x$  puis nous laissons une durée  $d$  s'écouler. Enfin, pour tout  $x \in X'$ , nous tirons la transition  $OK_{x > v(x)}$ . La configuration atteinte  $(M', \nu')$  vérifie alors  $(\ell, \nu + d) \mathcal{R}(M', \nu')$ .

*Second cas.* Simulation d'une transition discrète  $(\ell, \nu) \xrightarrow{a} (\ell', \nu')$ . Nous « exécutons » simplement le réseau de simulation associé à la transition discrète  $e$  correspondante : nous tirons la transition  $Tir_e$  et pour chaque mise à jour d'une horloge  $x$  (en suivant l'ordre défini par le réseau), nous appliquons le réseau qui vide le module d'horloge associé à  $x$ , puis nous le marquons à nouveau de façon appropriée. Ensuite, nous mettons à jour les places liées aux gardes diagonales pour lesquelles  $x$  intervient. Enfin, nous marquons la place  $\ell'$  et, pour chaque mise à jour d'une horloge  $x$ , les places  $Avant_{c_1(x)}^x$  où  $c_1(x)$  dénote la plus petite constante associée à  $x$ . Cette configuration  $(M', \nu')$  vérifie alors  $(\ell', \nu') \mathcal{R}(M', \nu')$ .

Enfin, il est facile de vérifier que cette simulation préserve les places finales et répétées.

**Preuve de  $\mathcal{L}(\mathcal{N}) \subseteq \mathcal{L}(\mathcal{A})$ .** Soit  $(M, \nu)$  une configuration accessible du réseau. Notons que nous avons  $\sum_{\ell \in L} M(\ell) \leq 1$ . Une configuration telle que  $\sum_{\ell \in L} M(\ell) = 1$  sera dite *tangible* et autrement elle sera dite *évanescente*. Étant donnée une configuration évanescente  $(M, \nu)$ ,  $(M', \nu')$  sera appelée *successeur tangible* de  $(M, \nu)$  si et seulement si elle est la première configuration tangible atteinte par une certaine séquence de tirs issue de  $(M, \nu)$ . Notons que les seules différences entre deux successeurs tangibles  $(M', \nu')$  et  $(M'', \nu'')$  d'une configuration évanescente  $(M, \nu)$  sont les suivantes : une transition  $Atteint_c^x$ ,  $Sort_c^x$ ,  $OK_{x > c}$  ou  $KO_{x < c}$  est franchissable dans un marquage et vient juste d'être tirée dans l'autre.

Nous définissons alors une relation  $\mathcal{R}'$  entre les configurations de l'automate et celles du réseau comme suit :  $(\ell, \nu) \mathcal{R}'(M, \nu)$  si et seulement si

- ou bien  $(M, \nu)$  est tangible et les conditions suivantes sont satisfaites. Nous donnons des conditions pour les différentes places :
  - $M(\ell) = 1$ ,
  - si  $M(\acute{E}gal_c^x) = 1$  alors  $v(x) = c$ ,

- si  $M(\text{Avant}_c^x) = 1$  alors  $v(x) = c' + \nu(\text{Atteint}_c^x)$  où  $c'$  est la constante précédent  $c$  (dans la liste triée des constantes liées à  $x$ ) ou 0 si  $c$  est la première de cette liste,
  - si  $M(\text{Après}_c^x) = 1 \wedge M(I_{x>c}) = 1$  alors  $v(x) = c + \nu(\text{OK}_{x>c})$ , et
  - si  $M(\text{Après}_c^x) = 1 \wedge M(T_{x>c}) = 1$  alors  $v(x) > c$ .
- ou bien  $(M, \nu)$  est évanescence et  $(\ell, v)\mathcal{R}'(M', \nu')$  pour une certaine configuration tangible  $(M', \nu')$  successeur de  $(M, \nu)$ .

L'observation fondamentale (obtenue par induction) établit que si  $(M, \nu)$  est tangible,  $(\ell, v)\mathcal{R}'(M, \nu)$  et  $M(T_{\text{cond}}) = 1$  alors  $v \models \text{cond}$ .

Observons d'abord que  $(\ell_0, \mathbf{0})\mathcal{R}'(M_0, \nu_0)$ , et supposons que  $(\ell, v)\mathcal{R}'(M, \nu)$ .

*Premier cas.* Simulation d'une transition de délai  $(M, \nu) \xrightarrow{d} (M, \nu + d)$ . Alors  $(\ell, v) \xrightarrow{d} (\ell, v + d)$  et  $(\ell, v + d)\mathcal{R}'(M, \nu + d)$  car  $(M, \nu)$  est nécessairement une configuration tangible.

*Second cas.* Simulation d'une transition discrète  $(M, \nu) \xrightarrow{t} (M', \nu')$ . Si  $t$  n'est pas une transition  $\text{Tir}_e$ , alors  $(\ell, v)\mathcal{R}'(M', \nu')$ . Si  $t = \text{Tir}_e$  pour un certain  $e = (\ell, g, a, \mu, \ell')$ , alors la place  $\ell$  est marquée et pour toute place d'entrée  $T_{\text{cond}}$  de  $t$ ,  $v \models \text{cond}$  est vérifié. Ainsi,  $(\ell, v) \xrightarrow{e} (\ell', v')$  et  $(\ell', v')\mathcal{R}'(M', \nu')$  puisque  $(\ell', v')\mathcal{R}'(M'', \nu'')$  où cette dernière configuration a été obtenue en simulant la transition  $e$ , comme cela a été détaillé précédemment.

Enfin, il est facile de vérifier que cette simulation préserve les places finales et répétées.

Pour conclure, soulignons que cette transformation est correcte vis-à-vis des langages acceptés, mais ne l'est pas vis-à-vis de la bisimulation. Ceci est inévitable d'après les résultats rappelés dans l'introduction. Dans notre traduction, le fait que le réseau construit n'est pas bisimilaire à l'automate temporisé provient des différentes configurations tangibles codant une même configuration de l'automate temporisé. Par exemple, le réseau  $\mathcal{N}$  peut, lorsqu'une horloge  $x$  vaut une valeur  $c \in \mathcal{C}_{\mathcal{N}}(x)$ , franchir en temps nul la transition  $\text{Sort}_c^x$ . La configuration atteinte n'est pas bisimilaire à la configuration de l'automate temporisé car elle ne possède plus de jeton dans la place  $T_{x \leq c}$  alors que la contrainte  $x \leq c$  est vérifiée dans l'automate.

### 4.2.3 Résultats de complexité et de concision

**Proposition 4.1** (Des automates étendus aux réseaux de Petri temporels). *Soit  $\mathcal{A}$  un automate temporisé étendu, alors il existe un réseau de Petri temporel sauf  $\mathcal{N}$  équivalent à  $\mathcal{A}$  du point de vue du langage temporisé accepté. La taille de ce réseau de Petri temporel, et la complexité temporelle de la construction dépendent de la classe à laquelle l'automate  $\mathcal{A}$  appartient. Cette complexité est quadratique en général, et linéaire si  $\mathcal{A}$  est sans gardes diagonales, ou s'il ne fait pas intervenir de mises à jour intégrales.*

**Preuve.** Nous considérons un automate temporisé étendu  $\mathcal{A}$ . La taille du réseau de Petri temporel construit précédemment est la somme des tailles de tous les sous-réseaux que nous avons décrits. D'abord, nous utilisons exactement une place pour représenter chaque état de contrôle de l'automate temporisé  $\mathcal{A}$ . De plus, les sous-réseaux représentant les transitions de  $\mathcal{A}$  ont une taille linéaire dans la taille de la transition qu'ils codent (voir figure 4.5). Enfin, la somme des tailles de tous les sous-réseaux dépendant d'une horloge  $x$  (module d'horloge, sous-réseau vidant le module d'horloge, sous-réseau marquant ce même module, sous-réseau de contrainte diagonale) est linéaire dans le nombre  $N_{\text{atomique}}(x)$  de contraintes non diagonales atomiques qui doivent être encodées pour la simulation de  $\mathcal{A}$ . En effet, le module d'horloge (Figure 4.1), le module pour le vider (Figure 4.2) et le module pour le marquer (Figure 4.3) ont tous une taille linéaire dans  $N_{\text{atomique}}(x)$ . De plus, le sous-réseau codant une contrainte diagonale est de taille constante, mais peut apparaître un nombre linéaire de

fois dans  $N_{atomique}(x)$ . La taille totale de notre construction est donc linéaire dans le nombre  $N_{atomique}$  de contraintes non diagonales atomiques qui doivent être représentées dans le réseau. Comme cela a été justifié au début de la présentation de la construction, ce nombre est ou bien quadratique ou bien linéaire dans la taille de  $\mathcal{A}$ , selon que  $\mathcal{A}$  fait intervenir à la fois des contraintes diagonales et des mises à jour intégrales (cas quadratique) ou qu'au contraire seule une de ces deux composantes apparaît (cas linéaire). Ceci conclut cette preuve.  $\square$

Dans [BC07], il a été démontré que les automates temporisés utilisant des gardes diagonales (mais aussi les automates temporisés utilisant des mises à jour intégrales) sont exponentiellement plus concis que les automates temporisés classiques. Appliquant ce résultat de concision, et utilisant la linéarité de la construction présentée plus haut, nous obtenons le résultat suivant pour les réseaux de Petri temporels saufs :

**Corollaire 4.2** (Concision des réseaux de Petri temporels : une borne inférieure). *Il existe une famille de réseaux de Petri temporels saufs  $(\mathcal{N}_k)_{k \in \mathbb{N}}$  telle que la taille de  $\mathcal{N}_k$  est  $\mathcal{O}(k^2 \log(k))$  et telle que tout automate temporisé sans gardes diagonales et sans mises à jour intégrales  $\mathcal{A}_k$  équivalent à  $\mathcal{N}_k$  (du point de vue du langage de mots finis accepté) a une taille supérieure à  $2^k$ .*

**Preuve.** Le réseau de Petri temporel  $\mathcal{N}_k$  représenté sur la figure 4.6 accepte le langage temporisé  $\mathcal{L}_k = \{(a, t_i)_{1 \leq i \leq 2^k} \mid 0 < t_i < t_{i+1}\}$ . En utilisant une légère adaptation de [BC07], nous pouvons démontrer que ce langage requiert un nombre exponentiel d'états de contrôle pour être accepté par un automate temporisé classique. Cependant, il est reconnu par le réseau de Petri  $\mathcal{N}_k$  représenté sur la figure 4.6 dont la taille est dans  $\mathcal{O}(k^2 \log(k))$ . Ce réseau implémente d'une certaine façon l'incrémentement d'un compteur binaire (chaque ligne du réseau correspond à un bit : si le jeton est dans la place de gauche, le bit correspondant vaut 0, sinon, il vaut 1).  $\square$

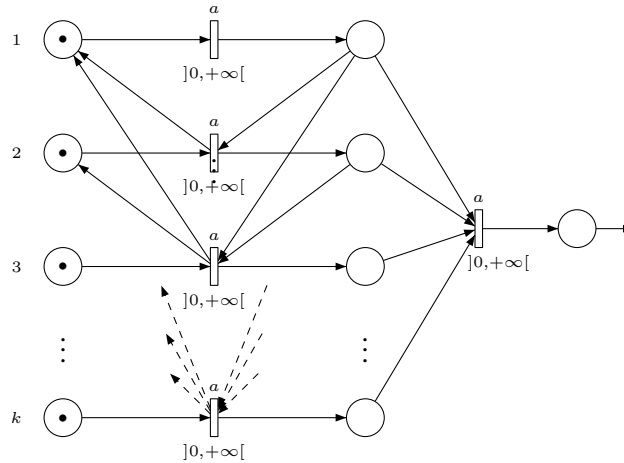


FIG. 4.6 – Le réseau de Petri temporel  $\mathcal{N}_k$

Enfin, ce résultat de concision est optimal puisqu'il existe également une borne supérieure exponentielle pour la taille de l'automate temporisé équivalent à un réseau de Petri temporel sauf.

**Proposition 4.3** (Concision des réseaux de Petri temporels : une borne supérieure [LR03]). *Soit  $\mathcal{N}$  un réseau de Petri temporel sauf, alors il existe un automate temporisé classique  $\mathcal{A}$  équivalent à  $\mathcal{N}$  (du point de vue des langages de mots finis) dont la taille est exponentielle dans la taille de  $\mathcal{N}$ .*

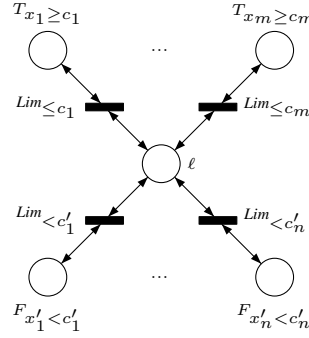


FIG. 4.7 – Traitement des invariants

### 4.3 Cas d'un réseau d'automates temporisés avec invariants

#### 4.3.1 Prise en compte des invariants

Dans la construction présentée précédemment, les invariants n'ont pas été pris en compte. En effet, nous avons montré dans le chapitre 1 sur les automates temporisés qu'il était possible de s'en passer pour des problèmes d'accessibilité, et donc pour des équivalences de langages. En revanche, nous avons montré que le retrait des invariants dans les réseaux d'automates temporisés est plus coûteux puisqu'il nécessite la construction de l'automate produit. Nous allons dans un premier temps expliquer comment la construction précédente peut être étendue de sorte à prendre en compte les invariants. Cette adaptation ne permet pas d'obtenir un résultat de bisimulation, mais seulement de contrôler les transitions de délai de sorte à vérifier les invariants, et ainsi satisfaire l'équivalence en termes de langages acceptés. La construction correspondante est représentée sur la figure 4.7. Nous considérons un invariant sur l'état de contrôle  $\ell$  de la forme  $Inv(\ell) = x_1 \leq c_1 \wedge \dots \wedge x_m \leq c_m \wedge x'_1 < c'_1 \wedge \dots \wedge x'_n < c'_n$ . D'après les propriétés mises en œuvre dans la preuve de correction de la construction précédente, nous avons :

- dès que l'horloge  $x$  atteint la valeur  $c$ , la place  $T_{x \geq c}$  est marquée,
- pour tout  $\eta > 0$ , il existe une exécution qui marque la place  $F_{x < c}$  lorsque  $x$  vaut  $c - \eta$ .

Ceci nous permet d'utiliser les places  $T_{x_1 \geq c_1}, \dots, T_{x_m \geq c_m}, F_{x'_1 < c'_1}, \dots, F_{x'_n < c'_n}$  pour déclencher les transitions  $Lim$  dès qu'un invariant est sur le point d'être violé. Ces transitions, immédiates, bloquent l'écoulement du temps jusqu'à ce que l'horloge en question soit mise à jour (à une valeur inférieure), ou jusqu'à ce qu'un changement d'état de contrôle ait lieu.

#### 4.3.2 Synchronisation

Dans cette section, nous décrivons la construction d'un réseau de Petri temporel équivalent à une synchronisation d'automates temporisés, *i.e.* à un réseau d'automates temporisés, du point de vue du langage temporel accepté. La preuve de correction de cette construction n'est pas donnée ici mais repose sur les techniques de simulation développées dans la section précédente.

Nous supposons donné un réseau d'automates temporisés  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$ . Nous notons de plus  $\mathcal{N}_i$  le réseau de Petri temporel obtenu en appliquant la construction présentée précédemment à l'automate  $\mathcal{A}_i$ . Pour chaque règle de la fonction de synchronisation  $f$ , nous ajoutons un sous-réseau de sorte à effectuer la synchronisation des transitions correspondantes de réseaux de Petri temporels  $\mathcal{N}_i$ . Nous expliquons ici la construction représentée sur la figure 4.8. D'abord, pour tout  $1 \leq i \leq n$ , et pour

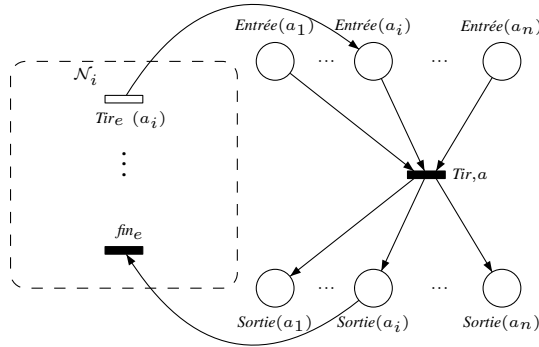


FIG. 4.8 – Simulation de la composition parallèle

toute lettre  $a_i$  apparaissant dans l'automate  $\mathcal{A}_i$ , nous ajoutons deux places  $Entrée(a_i)$  et  $Sortie(a_i)$ , que nous connectons aux transitions correspondantes  $Ttir_e$  (étiquetée par  $a_i$  dans  $\mathcal{N}_i$ ) et  $fin_e$  du réseau  $\mathcal{N}_i$ , pour toute transition  $e$  de  $\mathcal{A}_i$  étiquetée par  $a_i$ . Ensuite, pour toute règle  $f(a_1, \dots, a_i, \dots, a_n) = a$ , nous construisons une nouvelle transition immédiate  $Ttir$  étiquetée par  $a$  qui consomme toutes les places d'entrée  $Entrée(a_i)$  correspondant aux étiquettes  $a_1, \dots, a_n$  qui sont synchronisées, et produit en retour des jetons dans les places  $Sortie(a_i)$  correspondantes.

Il est aisé de vérifier que cette construction est linéaire dans la taille du réseau d'automates temporisés  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$ .

Enfin, notons que cette construction peut introduire des interblocages dans le réseau temporel construit. En effet, une action  $a_i$  peut être synchronisée avec d'autres actions de différentes manières. Cependant, ceci n'a pas d'effet sur le langage accepté.

## 4.4 Conclusion

Dans ces travaux, nous avons comparé les pouvoirs expressifs des réseaux de Petri temporels et de différentes extensions des automates temporisés. Plus précisément, nous avons présenté une transformation quadratique d'un automate temporisé avec gardes diagonales et mises à jour intégrales en un réseau de Petri temporel sauf. Cette transformation est même linéaire dès lors que les automates temporisés ne contiennent pas de gardes diagonales ou bien pas de mises à jour intégrales. Nous avons aussi obtenu que les réseaux de Petri temporels sont exponentiellement plus concis que les automates temporisés d'Alur et Dill, et nous avons exhibé une famille concrète de réseaux de Petri temporels qui témoigne de cette propriété de concision. Enfin, nous avons montré comment étendre cette transformation afin de traiter les invariants et les synchronisations d'automates, obtenant ainsi le cadre général des réseaux d'automates temporisés étendus avec invariants.

Au-delà des résultats théoriques obtenus qui donnent plus de détails sur les expressivités relatives des différents modèles, les transformations introduites suggèrent une méthode simple et unifiée pour l'analyse de réseaux d'automates temporisés étendus avec invariants. Cette méthode constitue une alternative aux algorithmes classiques pour l'analyse des automates temporisés implémentés dans les outils UPPAAL, HYTECH et KRONOS. Il s'agit dans un premier temps de traduire le réseau d'automates temporisés à analyser en un réseau de Petri temporel sauf à l'aide de notre traduction, puis dans un second temps d'utiliser les méthodes dédiées aux réseaux de Petri temporels comme la

construction du graphe des classes (implémentée dans l'outil TINA), la construction du graphe des zones (implémentée dans l'outil ROMEO) ou encore la construction de dépliages pour des objectifs comme le diagnostic [CJ06].

Afin d'évaluer cette perspective, nous avons donc développé un prototype permettant de traduire des automates temporisés, donnés selon la syntaxe d'UPPAAL, en des réseaux de Petri temporels analysables par TINA. Cependant, bien que linéaires, nos transformations font apparaître de nombreuses dépendances temporelles entre les différentes transitions du réseau, et l'efficacité de la méthode varie donc sensiblement. Ainsi, si l'automate possède de nombreuses horloges comparées à des valeurs différentes, cela va créer un nombre de classes important dans le graphe des classes construit par TINA. Au contraire, si les valeurs contre lesquelles sont testées les horloges sont peu nombreuses, ou si le nombre d'horloges est faible, alors le comportement est satisfaisant.

Une perspective intéressante consiste donc à chercher à caractériser plus précisément les classes d'automates temporisés pour lesquelles notre traduction donne des réseaux de Petri temporels sur lesquels le comportement de TINA ou de ROMEO est efficace.

D'autre part, il peut également être intéressant d'utiliser les techniques introduites dans ces travaux pour développer un nouvel algorithme pour l'analyse des réseaux d'automates temporisés, obtenu comme une adaptation de l'algorithme construisant le graphe des classes d'un réseau de Petri temporel. Ceci permettrait de s'affranchir de la première étape calculant explicitement le réseau de Petri temporel équivalent au réseau d'automates temporisés et ainsi de calculer « directement » une sorte de graphe des classes de ce réseau d'automates temporisés.

Il est également possible d'envisager une extension de nos traductions permettant d'exprimer d'autres extensions des automates temporisés, telles que celles introduites dans la section 1.5 ou dans [BDFP04].

Enfin, une dernière perspective consiste à s'intéresser à une autre extension des réseaux de Petri permettant d'exprimer des notions quantitatives de temps, les réseaux de Petri temporisés. C'est précisément l'objet du chapitre suivant.





## Chapitre 5

# Réseaux de Petri temporisés et automates temporisés

Dans ce chapitre, nous présentons une comparaison de ces deux modèles ayant pour but de mieux comprendre leurs fondements théoriques. Pour cela, nous réalisons une analyse précise de leurs expressivités relatives. L'essentiel des résultats présentés dans ce chapitre ont été publiés dans [BHR06b]. Une version longue de cet article est également disponible [BHR06c].

### 5.1 Introduction

Nous avons présenté dans les chapitres 1 et 2 les modèles des automates temporisés et des réseaux de Petri temporisés. Les automates temporisés constituent une extension simple et intuitive des automates finis et sont largement utilisés. Les réseaux de Petri temporisés sont quant à eux particulièrement adaptés aux systèmes infinis puisque nous avons vu que de nombreux problèmes sont décidables, même pour des réseaux non bornés.

En termes de comparaison de ces deux modèles, nous pouvons déjà affirmer que les réseaux de Petri temporisés (non bornés) ne peuvent pas toujours être transformés en automates temporisés car les langages non temporisés qu'ils acceptent peuvent ne pas être réguliers (voir exemple 2.4 page 55), alors que les langages non temporisés reconnus par des automates temporisés le sont toujours. Sur un autre aspect, il a souvent été affirmé que les réseaux de Petri temporisés sont plus expressifs que les automates temporisés puisqu'ils peuvent manipuler un nombre infini de jetons, chacun « possédant » son horloge. Dans les travaux présentés dans ce chapitre, nous nous sommes intéressés à une étude précise des expressivités relatives de ces deux modèles.

Suivant l'affirmation précédente, il semble naturel d'essayer de transformer les automates temporisés en réseaux de Petri temporisés bornés. Afin d'obtenir une transformation simple, il apparaît commode de modéliser chaque horloge de l'automate temporisé par une place du réseau de Petri temporisé. Cette transformation très simple fonctionne parfaitement pour la sous-classe des automates temporisés mettant systématiquement à jour la valeur des horloges qu'ils testent. Plus précisément, ceci signifie que si une garde de l'automate fait intervenir l'horloge  $x$ , alors cette horloge doit être mise à jour (ou simplement remise à zéro selon le modèle d'automates temporisés considéré) par cette même garde. La transformation correspondante est illustrée sur la figure 5.1.

Dans le cas général, la simulation des tests d'horloges qui ne sont pas mises à jour paraît plus délicate. Afin de conserver la simplicité et l'élégance de la transformation dans le cas général, il semble naturel d'étendre le modèle initial des réseaux de Petri temporisés avec des arcs de lecture, *i.e.*

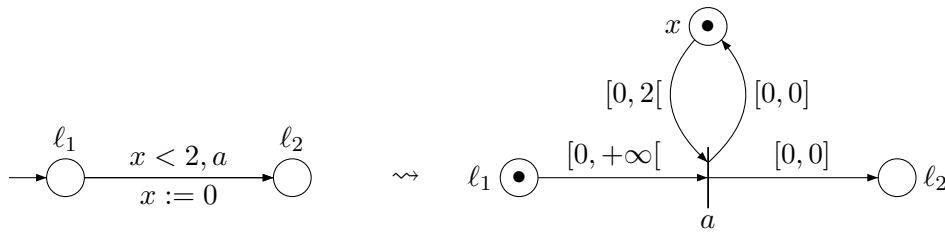


FIG. 5.1 – Cadre favorable à la transformation d’automates temporisés en réseaux de Petri temporisés.

des arcs qui testent la présence et l’âge de jetons, sans les consommer.

Les arcs de lecture ont été introduits dans le contexte non temporisé [MR95] afin de définir une sémantique de la concurrence plus raffinée pour les réseaux de Petri. Du point de vue de la sémantique des entrelacements, ils n’apportent pas d’expressivité puisqu’ils peuvent être simulés par deux arcs qui consomment et reproduisent le jeton. D’autre part, ces arcs ont déjà été introduits de façon indépendante dans les réseaux de Petri temporisés par Jiří Srba dans [Srb05] avec la même motivation : transformer les automates temporisés en réseaux de Petri temporisés. Cependant, dans ses travaux cet auteur ne s’intéresse pas aux questions d’expressivité liées à l’introduction des arcs de lecture.

Dans la section 5.2, une fois ce nouveau modèle introduit, nous allons montrer l’équivalence attendue entre les automates temporisés et les réseaux de Petri temporisés bornés avec arcs de lecture. Plusieurs questions naturelles se posent alors, auxquelles nous allons répondre dans la suite de ce chapitre :

- est-il nécessaire d’étendre le modèle des réseaux de Petri temporisés afin de pouvoir exprimer les automates temporisés ?
- les arcs de lecture augmentent-ils le pouvoir d’expression des réseaux de Petri temporisés ?
- que deviennent ces questions sous les contraintes de réseaux bornés, vis-à-vis des différentes équivalences de langages (mots finis, infinis, infinis non Zeno) introduites page 34 ?

Dans la section 5.3, nous établissons un résultat annexe (car il n’est pas relié à des problèmes d’expressivité) démontrant que le problème de couverture est décidable pour le modèle des réseaux de Petri temporisés avec arcs de lecture. Ensuite, nous présentons dans la section 5.4 des résultats préliminaires à l’étude précise de l’expressivité de ce modèle vis-à-vis de ses sous-classes. Ceci nous permet dans la section 5.5 de caractériser précisément le pouvoir d’expression des arcs de lecture. Nous nous intéressons ensuite dans la section 5.6 au pouvoir d’expression des mises à jour non déterministes. Ces différents résultats sont synthétisés, et exploités dans le cadre des automates temporisés dans la section 5.7. Enfin, nous donnons quelques conclusions et perspectives dans la section 5.8.

## 5.2 Réseaux de Petri temporisés avec arcs de lecture

Dans cette section, nous donnons une présentation formelle du modèle obtenu en ajoutant des arcs de lecture au modèle des réseaux de Petri temporisés présenté dans la section 2.3. Nous établissons ensuite une équivalence entre les réseaux bornés de cette classe et les automates temporisés.

### 5.2.1 Définition du modèle

Les définitions élémentaires ont déjà été données dans les chapitres 1 et 2.

Nous donnons ici la définition formelle d'un réseau de Petri temporisé avec arcs de lecture. Elle consiste en une simple extension de la définition 2.11 des réseaux de Petri temporisés, dans laquelle nous ajoutons une composante qui reflète les dépendances liées à ce nouveau type d'arc.

**Définition 5.1** (Réseau de Petri temporisé avec arcs de lecture). *Un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}$  est un uplet  $(P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  où :*

- $(P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  est un réseau de Petri temporisé, selon la définition 2.11, et
- $\circ(\cdot) \in \text{MEns}(P \times \mathcal{I})^T$  est l'application d'incidence de lecture (correspondant aux arcs de lecture).

La seule différence réside donc dans l'adjonction d'un nouveau type d'arc, étiqueté de la même façon par des multi-ensembles d'intervalles. Nous expliquons maintenant comment ils sont pris en compte dans la sémantique du réseau. Comme cela a été décrit informellement dans l'introduction, lors du tir d'une transition, la présence des jetons est testée, mais les jetons ne sont pas consommés.

**Définition 5.2** (Sémantique d'un réseau de Petri temporisé avec arcs de lecture). *Soit  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  un réseau de Petri temporel avec arcs de lecture. Sa sémantique est définie par le système de transitions temporisé  $\llbracket \mathcal{N} \rrbracket = (Q, q_0, \rightarrow)$  avec :*

- $Q = \text{MEns}(P \times \mathbb{T})$ ,
- $q_0 = \nu_0$ , où  $\nu_0(p) = M_0(p) \cdot (p, 0), \forall p \in P$ ,
- $\rightarrow$  définie pour  $\nu \in \text{MEns}(P \times \mathbb{T})$  par :
  - **transitions de délai** : pour  $d \in \mathbb{T}$ ,  $\nu \xrightarrow{d} \nu + d$ , où la somme est prise sur la composante de  $\mathbb{T}$ , i.e.  $(\nu + d)(p) = \nu(p) + d$  pour toute place  $p \in P$ .
  - **transitions discrètes** : pour  $a \in \Sigma_\varepsilon$ ,  $\nu \xrightarrow{a} \nu'$  si et seulement s'il existe un élément  $t \in T$  et des éléments  $\bullet\nu, \circ\nu, \nu^\bullet \in \text{MEns}(P \times \mathbb{R}_{\geq 0})$  tels que :

$$\left\{ \begin{array}{l} \Lambda(t) = a \\ \bullet\nu \models \bullet t \\ \circ\nu \models \circ t \\ \nu^\bullet \models t^\bullet \\ \bullet\nu + \circ\nu \leq \nu \\ \nu' = \nu - \bullet\nu + \nu^\bullet \end{array} \right.$$

Enfin, pour définir les exécutions acceptantes, nous considérons une condition d'acceptation de Büchi généralisé  $\text{Acc} = \{\text{acc}_1, \dots, \text{acc}_k\}$ . L'ensemble des configurations finales de  $\llbracket \mathcal{N} \rrbracket$ , noté  $Q_f$ , est défini par :

$$Q_F = \{\nu \in \text{MEns}(P \times \mathbb{T}) \mid \exists \text{acc} \in \text{Acc} \mid \pi_1(\nu) \text{ satisfait acc}\}$$

et la condition de Büchi généralisée associée à  $\llbracket \mathcal{N} \rrbracket$ , notée  $\{Q_r^1, \dots, Q_r^p\}$ , par :

$$Q_r^i = \{\nu \in \text{MEns}(P \times \mathbb{T}) \mid \pi_1(\nu) \text{ satisfait acc}_i\}$$

À nouveau la seule différence réside dans le traitement des arcs de lecture pour les transitions discrètes. Afin de pouvoir tirer une telle transition discrète, il faut pouvoir trouver dans le marquage courant  $\nu$ , en plus d'un sous-ensemble  $\bullet\nu$  de jetons qui vont être consommés, un autre sous-ensemble  $\circ\nu$  de jetons qui sont lus ( $\bullet\nu$  et  $\circ\nu$  sont disjoints), et qui seront donc laissés inchangés lors du tir (cf  $\nu' = \nu - \bullet\nu + \nu^\bullet$ ). Ces jetons lus doivent, de la même façon que les jetons consommés et produits, satisfaire certaines contraintes temporelles indiquées par des intervalles de temps. Cette sémantique est illustrée à l'aide de l'exemple 5.3 donné ci-après.

$$\text{Acc} = \{\text{actif} = 0, \text{inactif} = 0\}$$

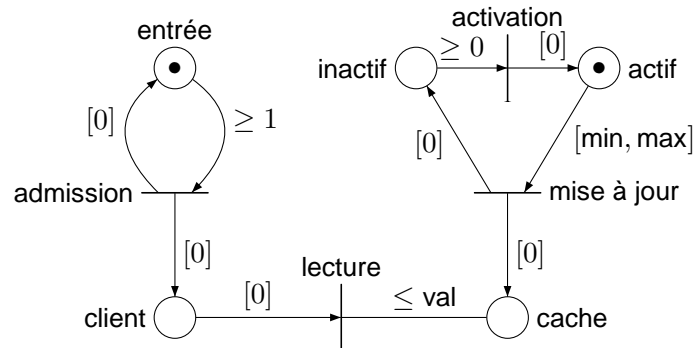


FIG. 5.2 – Un exemple de réseau de Petri temporisé avec arcs de lecture.

*Notations.* Nous utilisons les notations introduites pour la représentation des réseaux de Petri temporisés. Les arcs de lecture sont quant à eux représentés par des arcs non orientés.

**Exemple 5.3.** Un exemple de réseau de Petri temporisé avec arcs de lecture est représenté sur la figure 5.2. Ce réseau modélise une information fournie par un serveur et consultée de façon asynchrone par des clients (transition lecture). Comme cette information peut devenir obsolète au-delà d'une certaine durée de validité  $\text{val}$ , le serveur peut mettre à jour périodiquement le contenu de l'information (transition mise à jour), mais la fréquence de cette opération peut varier en fonction de la charge de travail du serveur. Le contrôle des entrées garantit qu'au moins une unité de temps s'écoule entre l'arrivée de deux clients (transition admission). Soulignons l'intérêt de l'arc de lecture entre les places cache et lecture : lorsqu'une transition lecture est franchie, un jeton d'âge 0 est consommé dans la place client et la présence dans la place cache d'au moins un jeton d'âge inférieur ou égal à  $\text{val}$  est testée. Cependant, ce jeton n'est pas consommé et son âge n'est pas modifié : il peut donc être réutilisé par la suite, ce qui reflète la notion de durée de validité évoquée plus haut. Insistons enfin sur le fait que la sémantique des réseaux de Petri temporisés (avec ou sans arcs de lecture) n'est pas « urgente » au sens où elle n'impose pas le franchissement des transitions. Par exemple, rien n'empêche le serveur de ne pas mettre à jour la valeur à temps et ainsi d'être bloqué à jamais (lorsque l'âge du jeton de la place actif est supérieur à  $\text{max}$ ). Une condition d'acceptation appropriée comme  $\text{Acc} = \{\text{actif} = 0, \text{inactif} = 0\}$  permet d'éviter un tel blocage en forçant le franchissement des transitions mise à jour et activation infiniment souvent.

Nous donnons un exemple de chemin dans ce réseau, en prenant  $\text{min} = 2$ ,  $\text{max} = 4$ , et  $\text{val} = 3$ .

$$\begin{array}{lcl}
 (\text{entrée}, 0) + (\text{actif}, 0) & \xrightarrow{(2)} & (\text{entrée}, 2) + (\text{actif}, 2) \\
 \text{mise à jour} & \xrightarrow{\quad} & (\text{entrée}, 2) + (\text{inactif}, 0) + (\text{cache}, 0) \\
 & \xrightarrow{(3)} & (\text{entrée}, 5) + (\text{inactif}, 3) + (\text{cache}, 3) \\
 \text{admission} & \xrightarrow{\quad} & (\text{entrée}, 0) + (\text{client}, 0) + (\text{inactif}, 3) + (\text{cache}, 3) \\
 \text{lecture} & \xrightarrow{\quad} & (\text{entrée}, 0) + (\text{inactif}, 3) + (\text{cache}, 3)
 \end{array}$$

┘

**Sous-classes des réseaux de Petri temporisés avec arcs de lecture.** Comme nous l'avons fait pour les précédents modèles que nous avons introduits, nous définissons un certain nombre de sous-classes du modèle des réseaux de Petri temporisés avec arcs de lecture. Certaines sont identiques à celles déjà introduites plus haut, mais nous redonnons l'ensemble des définitions car ces différentes sous-classes sont fondamentales pour l'étude du modèle que nous allons réaliser dans ce chapitre.

**Définition 5.4** (Sous-classes des réseaux de Petri temporisés avec arcs de lecture). *Soit  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  un réseau de Petri temporisé avec arcs de lecture. Celui-ci est*

- un réseau de Petri temporisé si pour toute transition  $t \in T$ ,  $\text{taille}(\circ t) = 0$ ,
- intégral si tous les intervalles apparaissant dans les multi-ensembles de  $\mathcal{N}$  sont éléments de  $\mathcal{I}_{\mathbb{N}}$ ,
- à remises à zéro si pour toute transition  $t \in T$ , et pour toute place  $p \in P$ ,  $I \neq [0, 0] \Rightarrow I \notin \text{dom}(t^\bullet(p))$ ,
- $k$ -borné si toutes les configurations  $\nu$  apparaissant le long d'une séquence de tir de  $\mathcal{N}$  sont telles que pour toute place  $p \in P$ ,  $\text{taille}(\nu(p)) \leq k$ ,
- borné s'il existe un entier naturel  $k \in \mathbb{N}$  pour lequel  $\mathcal{N}$  est  $k$ -borné,
- sauf s'il est 1-borné.

Toutes ces notions sont relativement classiques, mise à part la propriété de remises à zéro, qui correspond d'une certaine façon aux mises à jour standard des automates temporisés, qui n'autorisent que 0 comme nouvelle valeur des âges des jetons produits.

Notons que le réseau de Petri temporisé avec arcs de lecture de l'exemple 5.3 est intégral, à remises à zéro mais n'est pas borné car un nombre quelconque de jetons peut être produit dans les places cache et client.

### 5.2.2 Équivalence avec les automates temporisés

**Restrictions.** Parce que la sémantique des réseaux de Petri temporisés est dépourvue d'urgence et car nous démontrons des résultats de bisimilarité, nous considérons ici uniquement des automates temporisés sans invariants.

Nous établissons ici l'équivalence (en termes de langages) entre automates temporisés et réseaux de Petri temporisés avec arcs de lecture bornés. Le résultat suivant a été obtenu par Jiří Srba :

**Théorème 5.5** ([Srb05]). *Les réseaux de Petri temporisés saufs, avec arcs de lecture et restreints aux remises à zéro sont bisimilaires aux automates temporisés.*

Nous améliorons le résultat précédent en l'étendant à la classe des réseaux de Petri temporisés bornés avec arcs de lecture.

**Théorème 5.6.** *Les réseaux de Petri temporisés bornés avec arcs de lecture sont bisimilaires aux automates temporisés à mises à jour non déterministes. De plus, restreindre chacune des deux classes aux remises à zéro préserve l'équivalence.*

Au lieu de présenter une preuve utilisant le théorème 5.5, il nous est plus commode de présenter une preuve complète du théorème 5.6.

**Preuve.** Cette preuve procède en trois étapes. Dans un premier temps, nous démontrons le résultat esquissé dans l'introduction : étant donné un automate temporisé, il est possible de construire un réseau de Petri temporisé saufs avec arcs de lecture qui lui est bisimilaire. Ensuite, nous présentons une construction permettant de passer du cas borné au cas saufs pour les réseaux de Petri temporisés avec

arcs de lecture. Enfin, nous montrons comment transformer un réseau de Petri temporisé sauf avec arcs de lecture en un automate temporisé.

**Des automates temporisés aux réseaux de Petri temporisés saufs avec arcs de lecture.** Soit  $\mathcal{A} = (\Sigma_\varepsilon, X, L, T_{\mathcal{A}}, \ell_0, S)^1$  un automate temporisé (éventuellement avec mises à jour non déterministes). Nous construisons un réseau de Petri temporisé sauf (et donc a fortiori borné) avec arcs de lecture  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  de condition d'acceptation ACC comme suit.

- $P = L \cup X$ ,
- $M_0 = \ell_0 + X$ ,
- $T = T_{\mathcal{A}}$ ,
- pour toute transition  $t = \ell \xrightarrow{g, a, \mu} \ell'$  élément de  $T_{\mathcal{A}}$ ,
  - si  $x$  est telle que  $\mu(x)$  est défini, alors  $t^\bullet(x) = \mu(x)$ ,  $\bullet t(x) = g|_x$ , où  $g|_x$  est l'intervalle imposé à  $x$  par la contrainte  $g$ ,
  - si  $x$  est telle que  $\mu(x)$  n'est pas défini, alors  $\circ t(x) = g|_x$ ,
  - $\bullet t(\ell) = \mathbb{R}_{\geq 0}$
  - $t^\bullet(\ell') = [0]$ ,
  - $\Lambda(t) = a$ ,
- si  $\mathcal{S} = \{S_1, \dots, S_k\}$ , alors ACC est l'ensemble  $\{\text{acc}_1, \dots, \text{acc}_k\}$  où, pour tout  $1 \leq i \leq k$ ,  $\text{acc}_i = \sum_{\ell \in S_i} \ell = 1$

Le réseau  $\mathcal{N}$  que nous avons ainsi construit est fortement bisimilaire à l'automate temporisé original. En effet, considérons la relation  $\mathcal{R}$  définie par :

$$(\ell, v) \mathcal{R} \nu \text{ ssi } \begin{cases} \text{taille}(\nu(\ell)) = 1 \\ \text{taille}(\nu(\ell')) = 0 \quad \forall \ell' \neq \ell \\ \nu(x) = v(x) \quad \forall x \in X, \end{cases}$$

où  $(\ell, v) \in L \times \mathbb{T}^X$  est une configuration de  $\mathcal{A}$  et  $\nu \in \text{MEns}(\mathbb{T})^P$  est une configuration de  $\mathcal{N}$ . Il est facile de vérifier que  $\mathcal{R}$  est une relation de bisimulation forte qui respecte les configurations initiales et les conditions d'acceptation.

Enfin, notons qu'il y a exactement un jeton dans l'ensemble des places  $\ell$ , pour  $\ell \in L$  et exactement un jeton dans chaque place  $x$  pour  $x \in X$ . Le réseau construit ainsi est sauf (et donc borné) et ce réseau utilise des mises à jour non déterministes si et seulement si l'automate temporisé en contient. Cette construction est illustrée sur l'exemple 5.4.

**Réseaux de Petri temporisés avec arcs de lecture : des réseaux bornés aux réseaux saufs.**

Soit  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  un réseau de Petri temporisé borné avec arcs de lecture. Supposons de plus que ce réseau soit  $k$ -borné. Nous allons dans cette première étape construire un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}'$  bisimilaire et sauf par construction.

*Duplication des places.* Toute place  $p$  de  $\mathcal{N}$  est remplacée par  $2k$  places  $\{p_i^0, p_i^1 \mid 1 \leq i \leq k\}$  dans  $\mathcal{N}'$ . Pour tout indice  $i \in \{1, \dots, k\}$ , les places  $p_i^0$  et  $p_i^1$  seront mutuellement exclusives et les jetons de la place  $p$  dans  $\mathcal{N}$  (en nombre inférieur ou égal à  $k$ ) vont être redistribués dans les places  $p_i^1$ , pour  $i \in \{1, \dots, k\}$ . L'intuition de la construction est d'utiliser chacune de ces places pour simuler exactement un des au plus  $k$  jetons de  $p$ . Afin de garantir que le réseau est sauf, nous utilisons les places complémentaires  $p_i^0$ . Nous rendons ces deux places ( $p_i^0$  et  $p_i^1$ ) mutuellement exclusives en

<sup>1</sup>Remarquons que nous avons remplacé dans la définition les ensembles  $L_f$  et  $L_r$  d'états de contrôle finals et répétés par une condition de Büchi généralisée  $S$ . Ceci permet de faire coïncider les deux modèles.

imposant, lors d'une production (respectivement une consommation) d'un jeton dans la place  $p_i^1$ , de consommer (respectivement de produire) simultanément un jeton dans la place  $p_i^0$ .

*Duplication des transitions.* Fixons une transition  $t$  de  $\mathcal{N}$ . Cette transition va être remplacée dans  $\mathcal{N}'$  par plusieurs transitions.  $\bullet t(p)$  (respectivement  $\circ t(p)$ ,  $t^\bullet(p)$ ) est un multi-ensemble appartenant à  $\text{MEns}(\mathcal{T})$ , dont nous notons la taille  $s(p)$  (respectivement  $s'(p)$ ,  $s''(p)$ ). Nous ordonnons les jetons dans ces multi-ensembles et écrivons :

$$\begin{aligned}\bullet t(p) &= I_1 + \dots + I_{s(p)} \\ \circ t(p) &= I'_1 + \dots + I'_{s'(p)} \\ t^\bullet(p) &= I''_1 + \dots + I''_{s''(p)}\end{aligned}$$

Les dupliquées de  $t$  sont paramétrées par trois fonctions indiquant pour chaque place  $p$  dans quelles dupliquées de  $p$  les jetons doivent être consommés, lus et produits.

*Arc de consommation.* Soit  $p$  une place telle que  $s(p) > 0$ . Fixons une fonction injective  $\zeta_p$  définie de  $\{1, \dots, s(p)\}$  dans  $\mathbb{N}_k = \{1, \dots, k\}$ . Cette fonction détermine dans quelles places l'arc de consommation reliant  $t$  à  $p$  va consommer les  $s(p)$  jetons.

*Arc de lecture.* Soit  $p$  une place telle que  $s'(p) > 0$ . Fixons une fonction injective  $\zeta'_p$  définie de  $\{1, \dots, s'(p)\}$  dans  $\mathbb{N}_k$ . Cette fonction détermine dans quelles places l'arc de lecture reliant  $t$  à  $p$  va lire les  $s'(p)$  jetons.

*Arc de production.* Soit  $p$  une place telle que  $s''(p) > 0$ . Fixons une fonction injective  $\zeta''_p$  définie de  $\{1, \dots, s''(p)\}$  dans  $\mathbb{N}_k$ . Cette fonction détermine dans quelles places l'arc de production reliant  $t$  à  $p$  va produire les  $s''(p)$  jetons.

Nous définissons à présent la fonction  $\zeta$  (respectivement  $\zeta'$ ,  $\zeta''$ ) comme la fonction envoyant chaque place  $p \in P$  sur la fonction  $\zeta_p$  (respectivement sur  $\zeta'_p$ ,  $\zeta''_p$ ).

Supposons de plus que ces trois fonctions vérifient les conditions suivantes.

$$\forall p \in P, \begin{cases} \text{les images des applications } \zeta_p \text{ et } \zeta'_p \text{ sont disjointes,} \\ \text{les images des applications } \zeta'_p \text{ et } \zeta''_p \text{ sont disjointes.} \end{cases}$$

Ces conditions requièrent simplement que pour toute place  $p$ , la simulation de  $t$  ne cherche pas à lire et consommer un jeton dans une même dupliquée de  $p$ , ni à lire et produire un jeton dans une même dupliquée de  $p$ .

Pour tout triplet de fonctions  $(\zeta, \zeta', \zeta'')$  vérifiant les conditions précédentes, nous ajoutons à  $\mathcal{N}'$  la transition  $t' = t_{\zeta, \zeta', \zeta''}$  définie, pour tout place  $p \in P$ , par :

$$\begin{aligned}\forall i \in \{1, \dots, s(p)\}, & \begin{cases} \bullet t'(p_{\zeta_p^1(i)}^1) = I_i \\ t'^\bullet(p_{\zeta_p^0(i)}^0) = [0] \end{cases} \\ \forall j \in \{1, \dots, s'(p)\}, & \circ t'(p_{\zeta'_p^1(j)}^1) = I'_j \\ \forall l \in \{1, \dots, s''(p)\}, & \begin{cases} \bullet t'(p_{\zeta''_p^0(l)}^0) = \mathbb{R}_{\geq 0} \\ t'^\bullet(p_{\zeta''_p^1(l)}^1) = I''_l \end{cases}\end{aligned}$$

Enfin, l'étiquette de la transition  $t_{\zeta, \zeta', \zeta''}$  est simplement celle de  $t$ .

Étant donnée une place  $p \in P$ , les arcs reliant la transition  $t_{\zeta, \zeta', \zeta''}$  aux dupliquées de  $p$  sont représentés sur la figure 5.3.

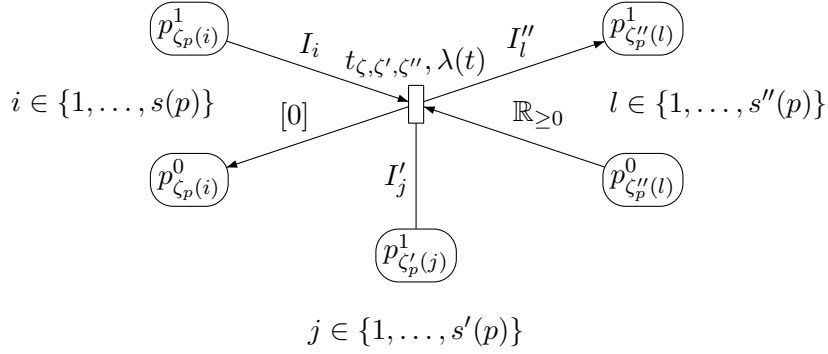


FIG. 5.3 – Réseaux de Petri temporisé avec arcs de lecture : du cas borné au cas sauf

*Marquage initial.* Étant donnée le marquage initial  $M_0 \in \text{MEns}(P)$  du réseau  $\mathcal{N}$ , nous définissons le marquage initial  $M'_0$  du réseau  $\mathcal{N}'$  par :

$$M'_0 = \sum_{p \in P} \sum_{i=1}^{M_0(p)} p_i^1.$$

*Conditions d'acceptation.* Enfin, la condition d'acceptation est modifiée de façon naturelle : toute occurrence d'une place  $p$  dans la condition d'acceptation est remplacée par le terme  $\sum_{i=1}^k (p_i^1)$ .

Il est relativement simple de vérifier que cette construction produit un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}'$  à la fois sauf et bisimilaire au réseau  $\mathcal{N}$ . Le fait que  $\mathcal{N}'$  soit sauf est évident par construction (rappelons que les places  $p_i^0$  et  $p_i^1$  sont complémentaires). L'existence d'une relation de bisimulation repose sur le fait qu'une configuration de  $\mathcal{N}$  ayant  $n$  jetons dans la place  $p$  est représentée dans  $\mathcal{N}'$  par une configuration dans laquelle  $n$  places  $p_i^1$  contiennent 1 jeton (et dont les âges correspondent aux jetons de  $p$ ) tandis que pour  $n - k$  autres indices  $i$ , la place  $p_i^0$  contient un jeton (d'âge quelconque). Il est alors facile de vérifier qu'une transition  $t$  est tirable dans  $\mathcal{N}$  depuis une configuration  $\nu$  si et seulement si une des dupliquées de  $t$  dans  $\mathcal{N}'$  est tirable depuis une des configurations de  $\mathcal{N}'$  correspondant à  $\nu$ . Puisque les marquages initiaux et les conditions d'acceptation sont préservés par cette relation de bisimulation, la bisimilarité implique les  $\{*, \omega, \omega_{nZ}\}$ -équivalences.

Enfin, notons également que le réseau obtenu est à remises à zéro si le réseau original l'est.

**Des réseaux de Petri temporisés saufs avec arcs de lecture aux automates temporisés.** Soit  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  un réseau de Petri temporisé sauf avec arcs de lecture, et une condition d'acceptation  $\text{Acc}$ . Nous allons définir un automate temporisé  $\mathcal{A} = (\Sigma_\varepsilon, X, L, T_{\mathcal{A}}, \ell_0, S)$  bisimilaire à  $\mathcal{N}$ . Afin de simplifier les notations de cette construction, nous noterons simplement  $\bullet t$  l'ensemble des places  $p \in P$  telles que  $\text{taille}(\bullet t(p)) > 0$  (et de même pour  $t^\bullet$  et  $\circ t$ ). Remarquons que comme  $\mathcal{N}$  est sauf, nous pouvons supposer que pour toute transition  $t \in T$ , nous avons  $\bullet t \cap \circ t = \emptyset$  et  $\circ t \cap t^\bullet = \emptyset$  (sinon la transition  $t$  ne pourra jamais être activée car le réseau est sauf).

Nous définissons alors  $\mathcal{A}$  comme suit :

- $X = \{x_p \mid p \in P\}$  ( $x_p$  représente l'horloge correspondant à la place  $p$ ),
- $L = 2^P$ ,
- il existe une transition  $\ell \xrightarrow{g, a, \mu} \ell' \in T_{\mathcal{A}}$  si et seulement s'il existe une transition  $t \in T$  dans  $\mathcal{N}$  telle que :
  - $\bullet t \cup \circ t \subseteq \ell$ ,



- $t^\bullet \cap (\ell \setminus \bullet t) = \emptyset$ ,
- $\ell' = (\ell \setminus \bullet t) \cup t^\bullet$ ,
- $g$  est la conjonction de toutes les contraintes  $x_p \in I_p$  telles que  $(p, I_p) \in \bullet t \cup \circ t$ ,
- $a = \Lambda(t)$ ,
- $\mu(x_p) = I_p$  si  $(p, I_p) \in t^\bullet$  et  $\mu(x_p) = \perp$  sinon.
- $\ell_0 = \text{dom}(M_0)$  (il y a exactement un jeton par place initialement marquée),
- si la condition d'acceptation de  $\mathcal{N}$  s'écrit  $\text{Acc} = \{\text{acc}_1, \dots, \text{acc}_k\}$ , alors nous définissons un ensemble  $S = \{S_1, \dots, S_k\}$  d'ensembles d'états de contrôle de  $\mathcal{A}$  par  $S_i = \{\ell \in L \mid \sum_{p \in \ell} p \models \text{acc}_i\}$ .

Remarquons que comme une place contient au plus un jeton, une seule horloge est suffisante pour coder le comportement de cette place. Il est facile de vérifier que cette construction est correcte. Enfin, l'automate temporisé met en jeu des mises à jour non déterministes si et seulement si le réseau de Petri temporisé n'est pas restreint aux remises à zéro.

Ceci conclut la preuve du théorème 5.6.  $\square$

**Exemple 5.7.** Nous illustrons la transformation d'un automate temporisé en un réseau de Petri temporisé sauf avec arcs de lecture sur la figure 5.4.  $\lrcorner$

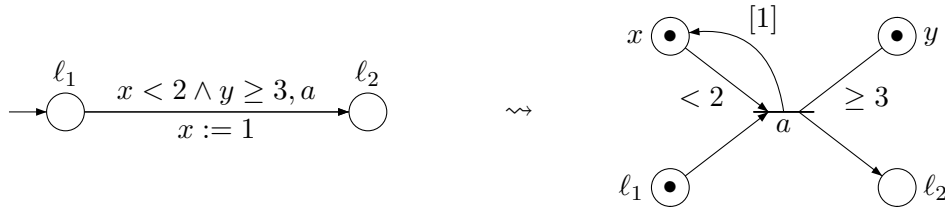


FIG. 5.4 – Un exemple de la transformation d'un automate temporisé en un réseau de Petri temporisé sauf avec arcs de lecture.

### 5.3 Problème de couverture

Nous démontrons à présent un résultat important pour cette nouvelle classe de modèles à savoir la décidabilité du problème de couverture.

Soit  $\mathcal{N}$  un réseau de Petri temporisé avec arcs de lecture de configuration initiale  $\nu_0$ . Soit  $N$  un ensemble de configurations de  $\mathcal{N}$ . Nous notons  $N^\uparrow$  la *fermeture par le haut* de  $N$ , i.e. l'ensemble de configurations  $\{\nu \mid \exists \nu' \in N, \nu' \leq \nu\}$ .

Le *problème de couverture* pour un réseau  $\mathcal{N}$  et un ensemble *fini* de configurations *rationnelles*  $N$  demande s'il existe un chemin dans  $\mathcal{N}$  issu de  $\nu_0$  atteignant une certaine configuration  $\nu \in N^\uparrow$ . Nous obtenons alors le résultat suivant :

**Théorème 5.8.** *Le problème de couverture est décidable pour les réseaux de Petri temporisés avec arcs de lecture.*

Ce résultat est à rapprocher du théorème 2.14 énonçant la décidabilité de ce problème pour les réseaux de Petri temporisés (sans arcs de lecture).

Afin de résoudre le problème de couverture pour les réseaux de Petri temporisés avec arcs de lecture, nous allons étendre la notion de régions, introduite par Alur et Dill pour l'analyse des automates temporisés (présentée dans la section 1.1) au cadre des réseaux de Petri temporisés avec arcs de lecture. Rappelons qu'une région est un objet mathématique permettant une manipulation symbolique des configurations du système. Cette notion a déjà été utilisée dans le cadre des réseaux de Petri temporisés (sans arcs de lecture) dans [Mah05], et a également été utilisée dans plusieurs autres contextes [OW04, OW05, LW05]. Une preuve basée sur l'utilisation de zones (définies dans le chapitre 6 pour les automates temporisés) pourrait également être envisagée, comme cela a été réalisé dans [AN01] dans le cadre des réseaux de Petri temporisés (sans arcs de lecture).

**Régions pour les réseaux de Petri temporisés avec arcs de lecture.** Soit  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  un réseau de Petri temporisé avec arcs de lecture, dans lequel les bornes des intervalles appartiennent à  $\mathbb{N} \cup \{+\infty\}$ . Soit  $N$  un ensemble fini de marquages dont les âges sont des entiers naturels. Il n'y a pas de perte de généralité à supposer les éléments précédents entiers. En effet, dans le cas contraire, nous pouvons raffiner la granularité des régions. Notons  $\max$  l'entier naturel le plus grand apparaissant dans les bornes des intervalles et dans les âges des jetons des configurations de  $N$ .

**Définition 5.9.** Une région  $\mathcal{R}$  de  $\mathcal{N}$  vis-à-vis de  $N$  est une séquence  $a_0 a_1 \dots a_n a_\infty$  dans laquelle  $n \in \mathbb{N}$  et pour tout  $0 \leq i \leq n$ ,  $a_i \in \text{MEns}(P \times \{0, 1, \dots, \max\})$  avec  $\text{taille}(a_i) \neq 0$  si  $i \neq 0$ , et  $a_\infty \in \text{MEns}(P \times \{\infty\})$ .

Dans un premier temps, nous expliquons de manière informelle la sémantique d'une région. Étant donné un multi-ensemble de jetons définissant une configuration, nous décrivons la région qui lui est associée. Nous plaçons dans  $a_\infty$  l'ensemble des jetons dont l'âge est strictement plus grand que la constante maximale  $\max$  et ne mémorisons pas leur âge. Nous plaçons ensuite dans  $a_0$  les autres jetons dont l'âge est entier (*i.e.* dont la partie fractionnaire est nulle) et indiquons leur âge. Enfin, nous ordonnons les jetons restants par ordre croissant de leur partie fractionnaire, les plaçons dans les multi-ensembles  $a_1, \dots, a_n$  (nous regroupons les jetons dont les parties fractionnaires sont égales) et ne retenons que la partie entière de leurs âges. Ainsi,  $n$  représente le nombre de valeurs fractionnaires différentes. Considérons par exemple la configuration  $(p, 1) + (p, 2.8) + (q, 0.8) + (q, 5.1) + (r, 1.5)$  et supposons que la constante maximale vaut 4. Alors la région contenant cette configuration est la région  $a_0 a_1 a_2 a_\infty$  où  $a_0 = (p, 1)$  (car il y a un unique jeton d'âge entier),  $a_\infty = (q, \infty)$  (car l'âge du jeton  $(q, 5.1)$  est 5.1, et se situe donc au-delà de la constante maximale),  $a_1 = (r, 1)$  (parmi les parties fractionnaires, 0.5 est la plus petite) et  $a_2 = (p, 2) + (q, 0)$  (tous les jetons dont l'âge a une partie fractionnaire égale à 0.8).

Nous définissons à présent formellement la sémantique des régions. Soit  $\phi$  l'application de  $\mathbb{T}$  vers  $\{0, 1, \dots, \max, \infty\}$  définie par : si  $x > \max$  alors  $\phi(x) = \infty$  sinon  $\phi(x) = \lfloor x \rfloor$ . Nous étendons  $\phi$  à  $P \times \mathbb{T}$  par  $\phi((p, x)) = (p, \phi(x))$  et à  $\text{MEns}(P \times \mathbb{T})$  par linéarité.

Soit  $\mathcal{R} = a_0 a_1 \dots a_n a_\infty$  une région, alors  $[\mathcal{R}]$  est l'ensemble des configurations  $\nu$  pour lesquelles il existe des configurations  $\nu_1, \nu_2, \dots, \nu_n, \nu_\infty \in \text{MEns}(P \times \mathbb{T})$  vérifiant :

- $\nu = a_0 + \nu_1 + \nu_2 + \dots + \nu_n + \nu_\infty$ ,
- $\forall 1 \leq i \leq n$ ,  $\phi(\nu_i) = a_i$  et  $\phi(\nu_\infty) = a_\infty$ ,
- $\forall 1 \leq i \leq n$ ,  $\forall (p, x) + (q, y) \leq \nu_i$ ,  $0 < x - \lfloor x \rfloor = y - \lfloor y \rfloor$ ,
- $\forall 1 \leq i < j \leq n$ ,  $\forall (p, x) \leq \nu_i$ ,  $(q, y) \leq \nu_j$ ,  $x - \lfloor x \rfloor < y - \lfloor y \rfloor$ .

Notons que chaque configuration  $\nu$  appartient à une unique région, nous notons celle-ci  $\mathcal{R}(\nu)$  et que si une configuration  $\nu \in \text{MEns}(P \times \mathbb{T})$  contient uniquement des jetons d'âge entier, alors  $[\mathcal{R}(\nu)] = \{\nu\}$ . C'est donc en particulier le cas pour les éléments de  $N$ , d'après l'hypothèse faite sur

$N$ . Le problème de couverture original de l'ensemble  $N$  est donc réduit au problème de couverture pour un ensemble fini de régions et donc au problème de couverture pour une unique région  $\mathcal{R}_f$ .

**Décidabilité du problème de couverture.** Avant de démontrer le théorème 5.8, nous énonçons le lemme suivant, dont la preuve est donnée après.

**Lemme 5.10.** *Soit  $\mathcal{R}$  une région et  $t$  une transition discrète. Nous avons alors :*

- (i) **Prédécesseurs temporels :** *l'ensemble des prédécesseurs temporels de  $[\mathcal{R}]^\uparrow$  est égal à une union finie  $\bigcup_{0 \leq i \leq k} [\mathcal{R}_i]^\uparrow$  où  $k \in \mathbb{N}$  et pour tout  $0 \leq i \leq k$ ,  $\mathcal{R}_i$  est une région. De plus,  $k$  et les régions  $\mathcal{R}_i$  sont calculables.*
- (ii) **Prédécesseurs discrets :** *l'ensemble des prédécesseurs discrets de  $[\mathcal{R}]^\uparrow$  par la transition  $t$  est égal à une union finie  $\bigcup_{0 \leq i \leq k} [\mathcal{R}_i]^\uparrow$  où  $k \in \mathbb{N}$  et pour tout  $0 \leq i \leq k$ ,  $\mathcal{R}_i$  est une région. De plus,  $k$  et les régions  $\mathcal{R}_i$  sont calculables.*

**Preuve du théorème 5.8.** Remarquons d'abord que, étant données deux régions  $\mathcal{R} = a_0 a_1 \dots a_n a_\infty$  et  $\mathcal{R}' = a'_0 a'_1 \dots a'_m a'_\infty$ , il est possible de tester si  $[\mathcal{R}]^\uparrow \subseteq [\mathcal{R}']^\uparrow$  : les conditions nécessaires et suffisantes sont  $a_0 \geq a'_0$ ,  $a_\infty \geq a'_\infty$  et l'existence d'une application strictement croissante  $\psi$  de  $\{1, \dots, n'\}$  dans  $\{1, \dots, n\}$  telle que pour tout  $1 \leq i \leq n'$ ,  $a_{\psi(i)} \geq a'_i$ .

Nous définissons un préordre  $\leq$  sur l'ensemble des régions par  $\mathcal{R} \leq \mathcal{R}'$  si et seulement si  $[\mathcal{R}']^\uparrow \subseteq [\mathcal{R}]^\uparrow$ . Alors, en utilisant le lemme d'Higman [Hig52], nous pouvons montrer qu'il forme un quasi-ordre bien fondé, *i.e.* pour toute séquence infinie de régions  $\{\mathcal{R}_i\}_{i \in \mathbb{N}}$ , il existe deux indices  $i < j$  tels que  $\mathcal{R}_i \leq \mathcal{R}_j$ . En effet, chaque région  $\mathcal{R}$  est une séquence finie de multi-ensembles sur un ensemble fini et donc, d'après [AN01, Théorème 1], le pré-ordre décrit plus haut est un quasi-ordre bien fondé.

L'algorithme pour résoudre le problème de couverture pour une région  $\mathcal{R}_f$  consiste alors à calculer de façon itérative les prédécesseurs (par écoulement du temps et par transitions discrètes) de  $[\mathcal{R}_f]^\uparrow$ . D'après le lemme 5.10, ces prédécesseurs sont toujours des unions finies de fermetures par le haut de régions. Nous stoppons alors l'exploration des prédécesseurs de la fermeture par le haut d'une région lorsque celle-ci est plus grande (pour le préordre  $\leq$ ) qu'une région déjà calculée. La correction de cette méthode découle de la remarque suivante : la relation  $\mathcal{R}_1 \leq \mathcal{R}_2$  entraîne que toutes les configurations accessibles depuis  $[\mathcal{R}_2]^\uparrow$  le sont également depuis  $[\mathcal{R}_1]^\uparrow$ . Démontrons à présent la terminaison de l'algorithme. Comme les prédécesseurs des ensembles  $[\mathcal{R}]^\uparrow$  ainsi calculés sont toujours des unions finies de fermetures par le haut de régions (d'après le lemme 5.10), le calcul réalisé par cet algorithme peut être vu comme un arbre à branchement fini. Supposons alors que cet algorithme ne termine pas, d'après le lemme de König cet arbre possède une branche infinie. Cependant, comme  $\leq$  est bien fondé, nous obtenons une région plus grande qu'une région déjà calculée et le calcul le long de cette branche doit donc être stoppé.

Ceci conclut la preuve du théorème 5.8. □

**Preuve du lemme 5.10.** Nous présentons ici comment calculer les prédécesseurs temporels et discrets de la fermeture par le haut d'une région  $\mathcal{R} = a_0 a_1 \dots a_n a_\infty$ .

**Prédécesseurs temporels.** Nous distinguons trois cas, selon la nature de la région  $\mathcal{R}$  :

Si  $a_0$  contient un jeton  $(p, 0)$ , alors il n'y a aucun prédécesseur temporel strict de  $[\mathcal{R}]^\uparrow$ .

Sinon, si  $\text{taille}(a_0) \neq 0$ , alors le prédécesseur temporel est  $[\mathcal{R}']^\uparrow$  avec  $\mathcal{R}' = a'_0 a_1 \dots a_n a'_{n+1} a_\infty$  où  $a'_0$  est le multi-ensemble vide et  $a'_{n+1}$  est obtenu à partir de  $a_0$  en décrémentant de 1 l'âge (intégral) de chaque jeton. De manière informelle, cette opération représente un écoulement du temps élémentaire inversé tel qu'aucun jeton de  $a_1$  n'atteint une valeur entière et aucun jeton de  $a_\infty$  n'atteint max.

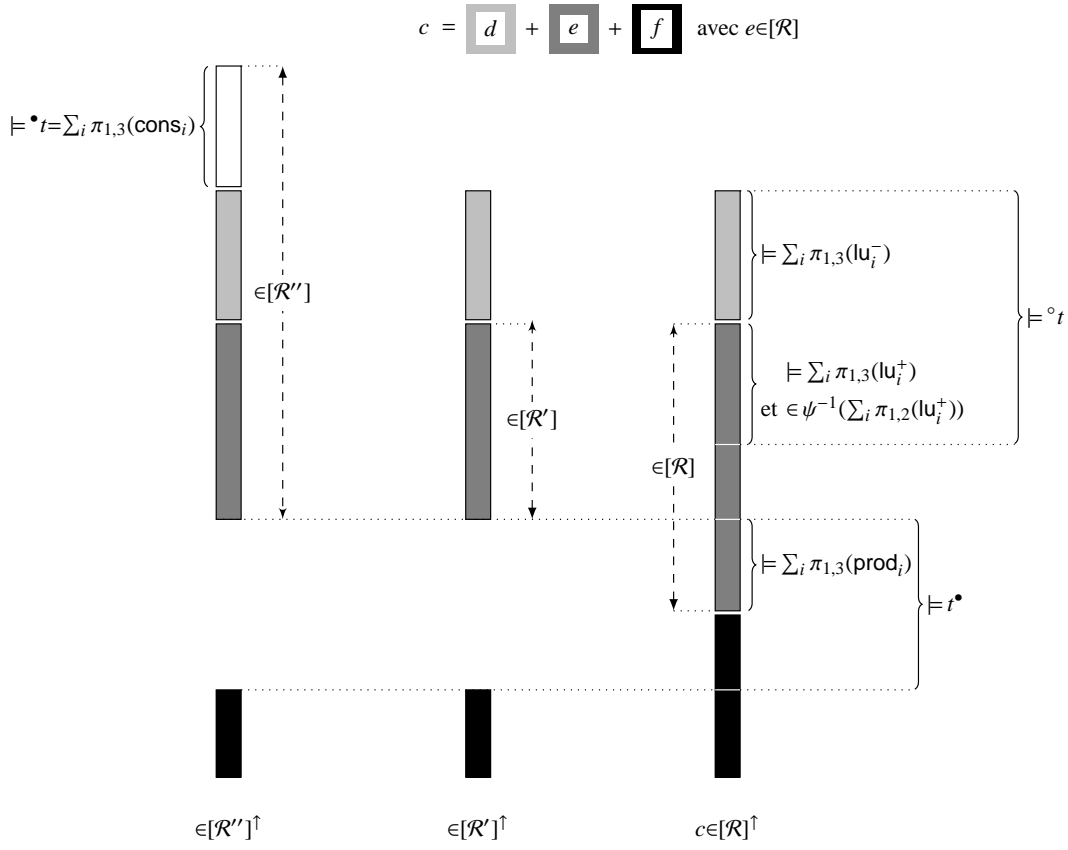


FIG. 5.5 – Décomposition de l'ensemble des jetons pour le calcul des prédécesseurs discrets.

Sinon, *i.e.* si  $\text{taille}(a_0) = 0$ , nous choisissons si les jetons de  $a_1$  vont en premier atteindre une valeur entière ou bien si ce sont en premier des jetons de  $a_\infty$  qui vont atteindre la valeur max. Il peut arriver que ce soit les jetons placés dans  $a_1$ , ou bien un certain sous-ensemble  $b_\infty \leq a_\infty$  des jetons placés  $a_\infty$ , ou encore les deux simultanément. Nous n'illustrons que ce dernier cas (qui implique  $n \geq 1$ ). Le prédécesseur temporel dans ce cas est alors l'ensemble  $[\mathcal{R}']^\uparrow$  où  $\mathcal{R}' = a'_0 a'_1 \dots a'_{n-1} a'_\infty$  est obtenu comme suit :

- $a'_\infty = a_\infty - b_\infty$ ,
- $a'_0 = a_1 + c_\infty$  où  $c_\infty$  est obtenu à partir de  $b_\infty$  en affectant l'âge de chaque jeton à max,
- $\forall 1 \leq i \leq n-1, a'_i = a_{i+1}$ .

**Prédécesseurs discrets.** Nous choisissons une transition  $t$ . Dans le reste de cette preuve, nous utilisons la notation suivante, définie pour tout entier naturel  $k \in \mathbb{N}$  :

$$\mathbb{N}_k^\infty = \{0, \dots, k\} \cup \{\infty\}$$

Remarquons qu'étant donné un intervalle  $I$  du réseau et un jeton  $(p, x)$  appartenant à un certain  $a_i$  pour  $i \in \mathbb{N}_n^\infty$ , nous pouvons déterminer si, étant donnée une configuration appartenant à cette région, le jeton correspondant appartient à  $I$ . Par propriété des régions, ceci est indépendant du choix de la configuration. Nous notons dans ce cas  $(i, x) \models I$ .

Nous considérons la fermeture par le haut de la région  $\mathcal{R} = a_0 a_1 \dots a_n a_\infty$  et voulons calculer l'ensemble de ses prédécesseurs par la transition  $t$ . La transition  $t$  produit un multi-ensemble de jetons

vérifiant la condition  $t^\bullet$ . Ces jetons peuvent être présents dans la région  $\mathcal{R}$ , mais ils peuvent aussi apparaître dans sa clôture par le haut. De façon similaire, certains jetons de  ${}^\circ t$  peuvent apparaître dans certains des  $a_i$ , mais ceci n'est pas nécessaire. Afin de décrire un ensemble de prédécesseurs possibles de  $[\mathcal{R}]^\dagger$ , nous choisissons des multi-ensembles de jetons  $\mathbf{prod}_i, \mathbf{lu}_i^+$  pour tout  $i \in \mathbb{N}_n^\infty$  tels que si  $i \neq \infty$ , ils sont éléments de  $\text{MEns}(P \times \{0, 1, \dots, \max\} \times \mathcal{I})$  et sinon ils sont éléments de  $\text{MEns}(P \times \{\infty\} \times \mathcal{I})$ . Nous supposons de plus qu'ils vérifient les conditions suivantes :

- pour tout  $i \in \mathbb{N}_n^\infty$ , pour tout  $(p, x, I) \leq \mathbf{prod}_i + \mathbf{lu}_i^+, (i, x) \models I$ ,
- pour tout  $i \in \mathbb{N}_n^\infty, \pi_{1,2}(\mathbf{prod}_i) + \pi_{1,2}(\mathbf{lu}_i^+) \leq a_i$ ,  
(rappelons que  $\pi_{1,2}$  projette les multi-ensembles sur les deux premières composantes – défini en section 2.1.)
- $\sum_{i \in \mathbb{N}_n^\infty} \pi_{1,3}(\mathbf{prod}_i) \leq t^\bullet$ ,
- $\sum_{i \in \mathbb{N}_n^\infty} \pi_{1,3}(\mathbf{lu}_i^+) \leq {}^\circ t$ .

Étant donné un indice  $i \in \mathbb{N}_n^\infty$ , le multi-ensemble  $\mathbf{prod}_i$  représente les jetons produits par  $t$  qui « appartiennent » à  $a_i$ , tandis que le multi-ensemble  $\mathbf{lu}_i^+$  représente les jetons lus par  $t$  et qui appartiennent à  $a_i$ . Cependant, il peut exister des jetons supplémentaires (qui sont lus ou produits) qui n'apparaissent pas dans les  $a_i$ . Soulignons à nouveau que ceci est dû au fait que nous considérons la fermeture par le haut de la région. Ceci explique que les parmi les conditions précédentes, les deux dernières conditions sont des inégalités. La figure 5.5 illustre cette décomposition.

Appliquant cette première décomposition, nous construisons la région intermédiaire  $\mathcal{R}' = a'_0 a'_1 \dots a'_n a'_\infty$  en soustrayant  $\pi_{1,2}(\mathbf{prod}_i)$  à  $a_i$  pour tout  $i$  et en retirant l'élément  $a_i - \pi_{1,2}(\mathbf{prod}_i)$  obtenu de la nouvelle séquence obtenue si sa taille est nulle. L'entier naturel  $n'$  est donc la taille de la nouvelle région obtenue ainsi.

Afin de compléter la simulation de la transition  $t$ , nous devons ajouter les jetons nécessaires aux arcs de lecture, ainsi que ceux qui sont consommés par  $t$ . Pour cela, étant donné un entier naturel  $n'' \in \mathbb{N}$ , nous définissons des multi-ensembles de jetons  $\mathbf{cons}_i, \mathbf{lu}_i^-$  pour tout  $i \in \mathbb{N}_{n''}^\infty$  tels que (comme plus haut) si  $i \neq \infty$ , ils sont éléments de  $\text{MEns}(P \times \{0, 1, \dots, \max\} \times \mathcal{I})$  et sinon ils sont éléments de  $\text{MEns}(P \times \{\infty\} \times \mathcal{I})$ . Nous supposons de plus qu'ils vérifient les conditions suivantes :

- pour tout  $i \in \mathbb{N}_{n''}^\infty$ , pour tout  $(p, x, I) \leq \mathbf{cons}_i + \mathbf{lu}_i^-, (i, x) \models I$ ,
- $\sum_{i \in \mathbb{N}_{n''}^\infty} \pi_{1,3}(\mathbf{cons}_i) = \bullet t$ ,
- $\sum_{i \in \mathbb{N}_{n''}^\infty} \pi_{1,3}(\mathbf{lu}_i^-) + \sum_{i \in \mathbb{N}_n^\infty} \pi_{1,3}(\mathbf{lu}_i^+) = {}^\circ t$ .

Les multi-ensembles  $\mathbf{lu}_i^-$  complètent les multi-ensembles  $\mathbf{lu}_i^+$  déjà introduits de sorte à vérifier la contrainte de lecture  ${}^\circ t$  tandis que les multi-ensembles  $\mathbf{cons}_i$  représentent les jetons requis par la contrainte  $\bullet t$ . Cette décomposition est illustrée sur la figure 5.5.

Étant donnée une application strictement croissante  $\psi$  de  $\{1, \dots, n'\}$  dans  $\{1, \dots, n''\}$ , nous définissons alors la région  $\mathcal{R}'' = a''_0 a''_1 \dots a''_n a''_\infty$  par :

- $a''_0 = a'_0 + \pi_{1,2}(\mathbf{cons}_0) + \pi_{1,2}(\mathbf{lu}_0^-)$ ,
- $a''_\infty = a'_\infty + \pi_{1,2}(\mathbf{cons}_\infty) + \pi_{1,2}(\mathbf{lu}_\infty^-)$ ,
- pour tout  $i \in \{1, \dots, n'\}$ , s'il existe un indice  $j \in \{1, \dots, n''\}$  tel que  $\psi(j) = i$  alors  $a''_i = a'_j + \pi_{1,2}(\mathbf{cons}_i) + \pi_{1,2}(\mathbf{lu}_i^-)$ , sinon  $a''_i = \pi_{1,2}(\mathbf{cons}_i) + \pi_{1,2}(\mathbf{lu}_i^-)$ .

Cette définition consiste simplement à ajouter à  $\mathcal{R}'$  les jetons correspondant aux arcs de lecture et aux arcs de consommation. Elle tient en plus compte du fait que les deux régions peuvent ne pas avoir la même taille (si  $n' \neq n''$ ) et fait donc intervenir la fonction  $\psi$ .

Sous ces conditions, nous pouvons finalement conclure que la région  $\mathcal{R}''$  obtenue est un prédécesseur par  $t$  de la région  $\mathcal{R}$  au sens suivant :

$$\forall \nu'' \in [\mathcal{R}'']^\dagger, \exists \nu \in [\mathcal{R}]^\dagger \text{ telle que } \nu'' \xrightarrow{t} \nu$$

Notons que la région  $\mathcal{R}''$  dépend des différents choix que nous avons réalisés (les multi-ensembles  $lu_i^+$ ,  $lu_i^-$ , etc., les entiers  $n'$ ,  $n''$  et l'application  $\psi$ ). Comme il existe un nombre fini de tels choix, nous obtenons que l'ensemble des prédécesseurs de  $[\mathcal{R}]^\uparrow$  par  $t$  peut être décrit comme une union finie d'ensembles  $[\mathcal{R}'']^\uparrow$ , ce que nous souhaitons démontrer.

Ceci conclut la preuve du lemme 5.10.  $\square$

## 5.4 Préliminaires à l'étude de l'expressivité

Dans cette section, nous établissons de premiers résultats cruciaux pour l'étude de l'expressivité menée dans la suite de ce chapitre. Tout d'abord, dans la sous-section 5.4.1, nous présentons deux langages temporisés qui permettent d'obtenir des inclusions strictes (en termes de pouvoir expressif) entre différentes classes. Ensuite, dans la sous-section 5.4.2, nous introduisons une notion de forme normale pour les réseaux de Petri avec arcs de lecture qui permet par la suite une étude simplifiée du modèle. Cette forme normale est utilisée intensivement dans les études menées dans les deux sections suivantes.

### 5.4.1 Deux langages temporisés discriminants

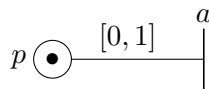
Nous introduisons ici deux langages temporisés qui permettent de distinguer plusieurs sous-classes des réseaux de Petri temporisés avec arcs de lecture. Il est important de remarquer que ces deux langages ne contiennent que des mots infinis Zeno. Cette remarque sera importante dans la suite de cette section.

**Le langage temporisé  $\mathcal{L}_1$ .** Le réseau  $\mathcal{N}_1$  représenté sur la figure 5.6(a) (avec une unique condition d'acceptation de Büchi  $p = 1$ ) est un réseau de Petri temporisé avec arcs de lecture à remises à zéro, intégral et borné qui reconnaît le langage temporisé

$$\mathcal{L}_1 = \{(a, \tau_1) \dots (a, \tau_n) \dots \mid 0 \leq \tau_1 \leq \dots \leq \tau_n \leq \dots \leq 1\}.$$

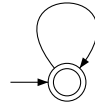
Remarquons que ce langage temporisé est également accepté par l'automate temporisé  $\mathcal{A}_1$  représenté sur la figure 5.6(b).

$$\text{Acc} = \{p = 1\}$$



(a) Le réseau  $\mathcal{N}_1$

$$x \leq 1, a$$



(b) L'automate temporisé  $\mathcal{A}_1$

FIG. 5.6 – Un langage temporisé  $\mathcal{L}_1$  reconnu par aucun réseau sans arcs de lecture.

**Lemme 5.11.** *Le langage temporisé  $\mathcal{L}_1$  n'est reconnu par aucun réseau de Petri temporisé sans arcs de lecture.*

**Preuve.** Supposons qu'il existe un réseau de Petri temporisé  $\mathcal{N}$  (sans arcs de lecture) acceptant le langage  $\mathcal{L}_1$ . Nous notons  $d$  le plus petit multiple commun des dénominateurs des constantes apparaissant dans les intervalles de  $\mathcal{N}$ . Choisissons un mot temporisé infini  $w = (a, \tau_1)(a, \tau_2) \dots (a, \tau_n) \dots$  tel que pour tout  $i \geq 1$ ,  $1 - \frac{1}{2d} < \tau_i < \tau_{i+1} < 1$ .

Le mot  $w$  est accepté par  $\mathcal{N}_1$ , et donc également par  $\mathcal{N}$  : il existe une séquence de tir infinie  $\sigma$  constituant une exécution acceptante pour  $w$  dans  $\mathcal{N}$ . Une telle séquence peut être décrite comme un élément de  $(T \times \mathbb{T})^*$  où  $T$  est l'ensemble des transitions du réseau  $\mathcal{N}$  et telle que la seconde composante est une séquence (croissante) de dates. Nous pouvons alors décomposer  $\sigma$  et écrire  $\sigma = \sigma_1(t_1, \tau_1)\sigma_2(t_2, \tau_2) \dots \sigma_n(t_n, \tau_n) \dots$  où pour tout entier naturel  $i \geq 1$ ,  $\sigma_i \in (T \times \mathbb{T})^*$  et de plus toutes les transitions de  $\sigma_i$  sont étiquetées par  $\varepsilon$  tandis que les transitions  $t_i$  sont étiquetées par  $a$ .

Notons *Jetons* l'ensemble des jetons faisant partie du marquage initial ou produits le long de la séquence  $\sigma_1$ . Cet ensemble est naturellement fini. En particulier, il existe un entier naturel  $n$  tel que les jetons de *Jetons* ne sont pas utilisés lors du tir de la séquence (infinie)  $(t_{n-1}, \tau_{n-1})\sigma_n(t_n, \tau_n) \dots$ . Considérons la sous-séquence de transitions temporisée  $(t_{n-1}, \tau_{n-1})\sigma_n(t_n, \tau_n)$ . Puisque  $\tau_{n-1} < \tau_n$ , il existe un suffixe  $(t'_0, \bar{\tau})(t'_1, \tau_n) \dots (t'_k, \tau_n)(t_n, \tau_n)$  de cette sous-séquence vérifiant  $\bar{\tau} < \tau_n$  ( $k$  peut éventuellement être nul). Notons  $\sigma'$  le préfixe fini de  $\sigma$  finissant par  $(t'_0, \bar{\tau})$  et écrivons  $\sigma = \sigma'\sigma''$ . Nous allons démontrer à présent que la séquence infinie  $\tilde{\sigma} = \sigma'(\sigma'' + \frac{1}{2d})$  peut être tirée dans  $\mathcal{N}$  ( $\sigma'' + \frac{1}{2d}$  est la séquence de transitions temporisée obtenue à partir de  $\sigma''$  en retardant les franchissements de transitions de  $\frac{1}{2d}$  unités de temps). Afin de démontrer ce résultat, allons nous analyser l'âge des jetons utilisés lors du tir d'une transition de  $\sigma'' = (t'_1, \tau_n) \dots (t'_k, \tau_n)(t_n, \tau_n)\sigma_{n+1}(t_{n+1}, \tau_{n+1}) \dots$  dans la séquence de transitions temporisée originale  $\sigma$  et nous allons montrer que (quand c'est nécessaire), nous pouvons modifier l'âge initial de ces jetons de sorte à rendre franchissable la séquence de transitions  $\tilde{\sigma}$ .

Choisissons un jeton dans la place  $p$  qui, le long de  $\sigma$ , est produit par une transition  $t$  et consommé lors du tir d'une transition  $t'$  dans le suffixe  $\sigma''$ . Ceci signifie en particulier que ce jeton n'est pas élément de *Jetons* et donc que la transition  $t$  a lieu à une date  $\tau$  vérifiant  $\tau_1 \leq \tau$ .

Si  $t$  est une transition de  $\sigma''$ , alors nous ne modifions pas l'âge initial de ce jeton dans  $\tilde{\sigma}$  puisque  $t$  et  $t'$  seront séparées par la même quantité de temps dans  $\sigma$  et dans  $\tilde{\sigma}$ , et le jeton pourra donc y être consommé de la même façon.

Si, au contraire,  $t$  intervient dans le préfixe  $\sigma'$  de  $\sigma$ , nous avons alors  $1 - 1/(2d) < \tau_1 \leq \tau \leq \bar{\tau} < \tau_n \leq \tau' < 1$  où  $\tau'$  est la date de franchissement de  $t'$  dans  $\sigma$ . Nous définissons  $\delta = \tau' - \tau$  : nous avons alors  $0 < \delta < \frac{1}{2d}$ . Dénotons par  $I^-$  l'intervalle de  $t^\bullet(p)$  associé à la production de ce jeton, et  $I^+$  l'intervalle de  $\bullet t'(p)$  associé à sa consommation. Notons tout d'abord que  $I^-$  et  $I^+$  ne peuvent pas être des singletons : supposons que ce soit le cas, alors  $I^- = [\frac{h}{d}, \frac{h}{d}]$  et  $I^+ = [\frac{k}{d}, \frac{k}{d}]$  avec  $h, k \in \mathbb{N}$ , mais alors  $\frac{k}{d} = \frac{h}{d} + \delta$ , ce qui est impossible puisque  $0 < \delta < \frac{1}{2d}$ .

- Supposons donc dans un premier temps que  $I^- = [\frac{h}{d}, \frac{h}{d}]$  et  $I^+ = (\frac{k}{d}, \frac{k'}{d})$  avec  $k < k'$  (les parenthèses définissant  $I^+$  peuvent être « strictes » ou « non-strictes »). L'âge du jeton lorsqu'il est consommé par  $t'$  dans  $\sigma$  est égal à  $\frac{h}{d} + \delta$  et est élément de  $I^+$ , nous avons donc  $h < k'$  et  $\frac{h}{d} + \delta + \frac{1}{2d} \in I^+$  (puisque  $0 < \delta < \frac{1}{2d}$ ). Dans ce cas, nous ne modifions pas l'âge initial du jeton pour franchir la séquence  $\tilde{\sigma}$  et le tir de  $t'$  peut être retardé de  $\frac{1}{2d}$  unités de temps.
- Le second cas correspond à supposer  $I^- = (\frac{h}{d}, \frac{h'}{d})$  et  $I^+ = [\frac{k}{d}, \frac{k}{d}]$  avec  $h < h'$ . L'âge du jeton lorsqu'il est produit dans  $\sigma$  (*i.e.* lorsque la transition  $t$  est tirée) est donc  $\frac{k}{d} - \delta \in I^-$ . Ainsi,  $h < k$  et  $\frac{k}{d} - \delta - \frac{1}{2d} \in I^-$  puisque  $0 < \delta < \frac{1}{2d}$ . Afin d'assurer le tir de la séquence  $\tilde{\sigma}$ , nous modifions donc l'âge initial du jeton lors de sa production par  $t$  dans cette séquence et lui affectons la valeur  $\frac{k}{d} - \delta - \frac{1}{2d}$  ce qui permet de retarder le franchissement de  $t'$  de  $\frac{1}{2d}$  unités de temps.
- Dans le dernier cas, nous supposons qu'à la fois  $I^-$  et  $I^+$  sont des intervalles non réduits à un singleton et donc sont de la forme  $I^- = (\frac{h}{d}, \frac{h'}{d})$  et  $I^+ = (\frac{k}{d}, \frac{k'}{d})$  avec  $h < h'$  et  $k < k'$ . Nous notons alors  $\alpha$  l'âge initial du jeton lors du tir de la transition  $t$  dans  $\sigma$  :  $\alpha + \delta (\leq \frac{k'}{d})$  est l'âge de ce jeton lors de sa consommation par  $t'$  dans  $\sigma$ . Deux cas sont à envisager : si

$\alpha + \delta < \frac{k'}{d} - \frac{1}{2d}$ , alors nous ne modifions pas son âge initial dans  $\tilde{\sigma}$  et le franchissement de  $t'$  peut être retardé de  $\frac{1}{2d}$  unités de temps. Si au contraire  $\alpha \geq \frac{k'}{d} - \frac{1}{2d} - \delta$  : alors  $\frac{k'-1}{d} < \alpha < \frac{k'}{d}$  et donc  $h \leq k' - 1 < h'$ . Dans  $\tilde{\sigma}$ , choisissons comme nouvel âge initial du jeton  $\alpha' = \frac{k'-1}{d} + \beta$  avec  $0 < \beta < \frac{1}{2d}$ . Nous pouvons alors vérifier que  $\alpha' \in I^-$  et  $\alpha' + \delta + \frac{1}{2d} \in I^+$ , ainsi le franchissement de  $t'$  peut être retardé de  $\frac{1}{2d}$  unités de temps.

Avec ces nouveaux âges initiaux pour les jetons, la séquence de transitions temporisée  $\tilde{\sigma}$  est franchissable et accepte le mot temporisé  $(a, \tau_1) \dots (a, \tau_{n-1}) (a, \tau_n + \frac{1}{2d}) (a, \tau_{n+1} + \frac{1}{2d}) \dots$ . De plus, les marquages discrets obtenus respectivement le long des exécutions acceptant le mot temporisé initial et le mot temporisé précédent sont identiques, ces deux mots temporisés sont donc acceptés par  $\mathcal{N}$ . Cependant le mot temporisé reconnu le long de  $\tilde{\sigma}$  ne devrait pas être accepté car il n'appartient pas au langage  $\mathcal{L}_1$  (du fait de  $\tau_n + \frac{1}{2d} > 1$ ). Ceci contredit donc l'hypothèse de l'existence d'un réseau de Petri temporisé sans arcs de lecture acceptant  $\mathcal{L}_1$  et conclut la preuve de ce lemme.  $\square$

**Le langage temporisé  $\mathcal{L}_2$ .** Le réseau  $\mathcal{N}_2$  représenté sur la figure 5.7(a) est un réseau de Petri temporisé avec arcs de lecture et mises à jour non déterministes, intégral et borné qui reconnaît le langage temporisé

$$\mathcal{L}_2 = \{(a, 0)(b, \tau_1) \dots (b, \tau_n) \dots \mid \exists \tau < 1 \text{ tel que } 0 \leq \tau_1 \leq \dots \leq \tau_n \leq \dots < \tau\}.$$

Remarquons également, ce sera utile dans la sous-section 5.7.2, que ce langage temporisé est également accepté par l'automate temporisé  $\mathcal{A}_2$  représenté sur la figure 5.7(b) (qui utilise une mise à jour non déterministe de l'horloge  $x$  dans l'intervalle  $]0, 1[$ ).

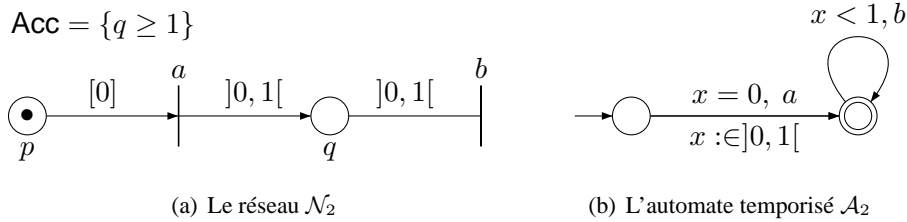


FIG. 5.7 – Le langage temporisé  $\mathcal{L}_2$  reconnu par aucun réseau intégral à remises à zéro.

**Lemme 5.12.** *Le langage temporisé  $\mathcal{L}_2$  n'est reconnu par aucun réseau de Petri temporisé à arcs de lecture qui soit intégral et à remises à zéro.*

**Preuve.** Nous procédons par l'absurde. Supposons qu'il existe un réseau de Petri temporisé  $\mathcal{N}$  avec arcs de lecture intégral, et n'utilisant que des remises à zéro, acceptant le langage  $\mathcal{L}_2$ . Choisissons un mot temporisé  $w = (a, 0) (b, \tau_1) \dots (b, \tau_i) \dots$  appartenant à  $\mathcal{L}_2$ , vérifiant  $0 < \tau_1 \leq \tau_2 \leq \dots \leq \tau_i \leq \dots < \tau$  et  $\lim_{i \rightarrow \infty} \tau_i = \tau$ . Nous notons  $\sigma \in (T \times \mathbb{T})^*$  une séquence de transitions temporisée de  $\mathcal{N}$  acceptant  $w$ .

Nous pouvons écrire  $\sigma = \sigma_1 \sigma_2$  avec  $\sigma_1$  une séquence de transitions temporisée instantanée et  $\sigma_2 = (t_0, d) \bar{\sigma}_2$  pour un certain délai  $d > 0$  (ainsi  $t_0$  est la première transition de  $\sigma$  franchie à une date strictement positive). Nous affirmons que  $\sigma' = \sigma_1 \sigma'_2$ , où  $\sigma'_2$  est obtenue à partir de  $\sigma_2$  en la retardant de  $1 - \tau$ , est une séquence de transitions de  $\mathcal{N}$ . Sélectionnons une occurrence d'une transition  $t$  tirée dans  $\sigma_2$  et un jeton lu ou consommé par  $t$ , dont l'intervalle associé est noté  $I$ . Si ce jeton a été produit par une transition appartenant à  $\sigma_2$ , alors il a le même âge dans  $\sigma'_2$ . Si au contraire ce jeton fait partie du marquage initial, ou a été produit le long de  $\sigma_1$ , alors son âge  $x$  lors du tir de  $t$  dans  $\sigma_2$  vérifie



$0 < d \leq x < \tau < 1$ , et donc  $]0, 1[ \subseteq I$  (car nous avons supposé que le réseau est intégral et à remises à zéro). L'âge de ce jeton lors du tir de  $t$  dans  $\sigma'_2$  est donc  $x + 1 - \tau$  et vérifie donc encore  $0 < x + 1 - \tau < 1$ . Ainsi, la même occurrence de  $t$  est franchissable dans  $\sigma'_2$ .

Comme les séquences non temporisées associées à  $\sigma$  et  $\sigma'$  sont égales,  $\sigma'$  est une séquence acceptable. Le mot temporisé associé à  $\sigma'$  est donc accepté et s'écrit  $w' = (a, 0)(b, \tau_1 + 1 - \tau) \dots (b, \tau_i + 1 - \tau) \dots$  avec  $\lim_{i \rightarrow \infty} \tau_i + 1 - \tau = 1$ . Ceci implique  $w' \notin \mathcal{L}_2$ , ce qui contredit l'hypothèse de l'existence de  $\mathcal{N}$ .

Finalement, il n'existe pas de réseau de Petri temporisé intégral avec arcs de lecture et remises à zéro qui accepte le langage  $\mathcal{L}_2$ .  $\square$

### 5.4.2 Forme normale pour les réseaux de Petri temporisés

Nous présentons une transformation des réseaux de Petri temporisés avec arcs de lecture préservant à la fois les langages de mots finis et infinis (Zeno et non Zeno), ainsi que les caractères borné et intégral des réseaux. Cette construction transforme le réseau en lui imposant de fortes contraintes syntaxiques sur les places, qui permettent de simplifier les études que nous allons mener par la suite. Cette transformation est décomposée en trois étapes. La première étape consiste à diviser les intervalles de sorte que deux intervalles soient toujours ou bien disjoints, ou bien égaux. La seconde étape est d'une certaine façon proche de la construction des régions à une dimension de [LMS04] et mémorise l'âge des jetons et comment le temps s'écoule. Enfin, la dernière étape duplique des places afin que tous les arcs de production (respectivement de consommation) connectés à une place soient étiquetés par le même intervalle.

**Proposition 5.13.** *Pour tout réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}$ , nous pouvons construire de façon effective un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}'$  qui est  $\{*, \omega_{nZ}, \omega\}$ -équivalent à  $\mathcal{N}$ , et dans lequel toutes les places  $p$  sont dans l'un des cinq motifs représentés sur la figure 5.8, qui se lit ainsi : « il existe un élément  $a \in \mathbb{Q}_{>0} \cup \{+\infty\}$  tel que tout arc connecté à cette place  $p$  soit étiqueté par un multi-ensemble dont la forme est spécifiée sur la figure, selon qu'il s'agit d'un arc de consommation, de consommation ou de production ». De plus, la construction préserve les caractères borné et intégral du réseau.*

**Remarque 5.14.** Le résultat de la proposition 5.13 n'impose pas que les différents arcs du même type aient exactement la même étiquette, mais seulement que leurs étiquettes aient la même forme. Plus précisément, seule la valeur de  $a$  est partagée. Ainsi, une place  $p$  reliée uniquement à deux arcs de production étiquetés respectivement par  $2 \cdot [0]$  et  $4 \cdot [0]$  est configurée selon le motif  $\mathcal{P}_1$ .  $\lrcorner$

Afin d'éviter les difficultés liées au marquage initial, nous effectuons d'abord une transformation évidente sur le réseau. Nous ajoutons une place  $p_{init}$  contenant un unique jeton, et une transition  $t_{init}$  étiquetée par  $\varepsilon$ , dont l'unique arc de consommation étiqueté par  $[0]$  est connecté à  $p_{init}$  et dont les arcs de production correspondent au marquage initial, i.e. pour tout  $p \in P$ ,  $t_{init} \bullet (p) = M_0(p) \cdot [0]$ . Toutes les autres places sont initialement vides. Enfin, nous ajoutons  $p_{init} = 0$  à la condition d'acceptation. Il est aisé de vérifier que cette transformation ne modifie pas le langage accepté. Dans la suite, nous supposons donc que nous avons déjà appliqué cette transformation au réseau, et nous appliquerons toutes les prochaines transformations à toutes les places exceptée la place  $p_{init}$ .

Comme annoncé précédemment, afin de démontrer la proposition 5.13, nous procédons en trois étapes, et construisons successivement un nouveau réseau qui satisfait les restrictions syntaxiques (1), (2) et (3) suivantes :

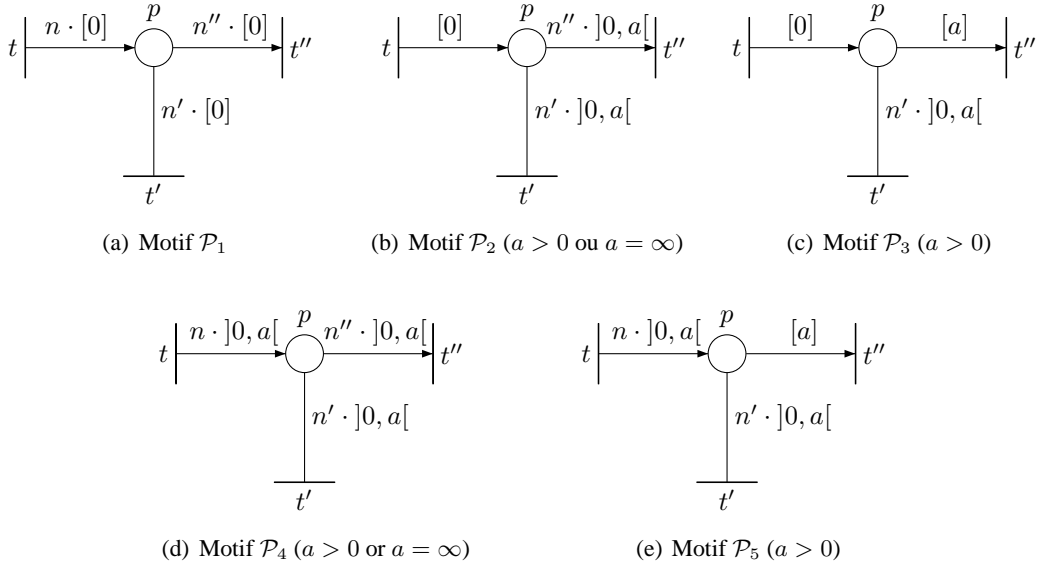


FIG. 5.8 – Normalisés d'un réseau de Petri temporisé avec arcs de lecture.

- (1) Pour chaque place, il existe un ensemble fini d'intervalles deux à deux disjoints  $\{I_k\}_{1 \leq k \leq K}$  tel que tout arc connecté à cette place possède un multi-ensemble de la forme  $\sum_{1 \leq k \leq K} n_k \cdot I_k$ . De plus, chaque  $I_k$  est ou bien  $]a]$  ou bien  $]a, b[$  avec  $a \in \mathbb{Q}_{\geq 0}$  et  $b \in \mathbb{Q}_{> 0} \cup \{+\infty\}$ .
- (2) Pour chaque place<sup>2</sup>,
  - (i) ou bien elle est connectée à (éventuellement) plusieurs arcs de production étiquetés par des multi-ensembles  $n \cdot ]0]$ , (éventuellement) plusieurs arcs de lecture étiquetés par des multi-ensembles  $n' \cdot ]0]$  et (éventuellement) plusieurs arcs de consommation étiquetés par des multi-ensembles  $n'' \cdot ]0]$ .
  - (ii) ou bien il existe un nombre rationnel  $a \in \mathbb{Q}_{> 0}$  tel que cette place est connectée à au plus un arc de production étiqueté par  $]0]$ , (éventuellement) plusieurs arcs de production étiquetés par des multi-ensembles  $n \cdot ]0, a[$ , (éventuellement) plusieurs arcs de lecture étiquetés par des multi-ensembles  $n' \cdot ]0, a[$ , un arc de consommation étiqueté par  $]a]$  et (éventuellement) plusieurs arcs de consommation étiquetés par des multi-ensembles  $n'' \cdot ]0, a[$ .
  - (iii) ou bien elle est connectée à un arc de production étiqueté par  $]0]$ , (éventuellement) plusieurs arcs de production étiquetés par des multi-ensembles  $n \cdot ]0, +\infty[$ , (éventuellement) plusieurs arcs de lecture étiquetés par des multi-ensembles  $n' \cdot ]0, +\infty[$  et (éventuellement) plusieurs arcs de consommation étiquetés par des multi-ensembles  $n'' \cdot ]0, +\infty[$ .
- (3) Toute place est configurée comme l'un des cinq motifs représentés sur la figure 5.8.

Dans tous les lemmes suivants, l'équivalence mentionnée est la  $\{*, \omega, \omega_{nZ}\}$ -équivalence, ce qui signifie que les constructions conservent les langages temporisés de mots finis et infinis.

Le processus global de construction procède ainsi : nous fixons un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}$  puis nous construisons successivement trois réseaux de Petri temporisés avec arcs de lecture  $\mathcal{N}_1, \mathcal{N}_2$  et  $\mathcal{N}_3$  obtenus respectivement par les lemmes 5.15, 5.16 et 5.18.

**Lemme 5.15.** *Nous pouvons construire un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}_1$ , équivalent à  $\mathcal{N}$ , et vérifiant la restriction (1).*

<sup>2</sup>Les paramètres  $n, n'$  et  $n''$  ne sont pas nécessairement partagés par les arcs.

**Preuve.** Soit  $p$  une place de  $\mathcal{N}$ . Nous considérons les bornes finies des intervalles apparaissant dans les multi-ensembles d'un arc connecté à  $p$ , disons  $\{a_1, \dots, a_m\}$  avec  $i < j \Rightarrow a_i < a_j$ . Nous définissons ensuite l'ensemble  $SI_p = \{[a_1, a_1], ]a_1, a_2[, \dots, ]a_{m-1}, a_m[, [a_m, a_m], ]a_m, +\infty[ \}$ . Sans perte de généralité, nous pouvons supposer que  $a_1 = 0$ . De plus, afin de faciliter la présentation, nous définissons  $a_{m+1} = +\infty$ , adoptons la convention  $a_{m+1} - a_m = +\infty$  et écrivons l'ensemble  $SI_p$  comme  $SI_p = \{I_k\}_{1 \leq k \leq K}$ . Remarquons que pour tout intervalle  $I_k \in SI_p$  et pour tout intervalle  $I$  apparaissant dans un multi-ensemble d'un arc connecté à  $p$ , nous avons ou bien  $I \cap I_k = \emptyset$  ou bien  $I \cap I_k = I_k$ .

Nous allons itérativement appliquer les transformations suivantes aux transitions connectées à  $p$ . Choisissons donc une transition  $t$  connectée à  $p$  par un arc dont le multi-ensemble associé s'écrit  $x = \sum_{1 \leq k' \leq K'} n_{k'} \cdot J_{k'}$ . Nous dupliquons la transition  $t$  avec les mêmes arcs et les mêmes multi-ensembles excepté l'arc concerné par la transformation. Nous notons une telle transition dupliquée  $t_\phi$ , où  $\phi$  est une application de  $\{1, \dots, K\} \times \{1, \dots, K'\}$  dans  $\mathbb{N}$  telle que  $I_k \cap J_{k'} = \emptyset \Rightarrow \phi(k, k') = 0$  et  $\sum_{1 \leq k \leq K} \phi(k, k') = n_{k'}$ . Le multi-ensemble modifié est défini par :

$$\begin{aligned} x_\phi &= \sum_{1 \leq k' \leq K'} \sum_{1 \leq k \leq K} \phi(k, k') \cdot (I_k \cap J_{k'}) \\ &= \sum_{1 \leq k' \leq K'} \sum_{1 \leq k \leq K} \phi(k, k') \cdot I_k \\ &= \sum_{1 \leq k \leq K} \left( \sum_{1 \leq k' \leq K'} \phi(k, k') \right) \cdot I_k. \end{aligned}$$

Cette transformation est correcte. En effet, étant donné n'importe quel choix d'un élément  $b \in \text{MEns}(\mathbb{T} \times \mathcal{I})$  avec  $\pi_2(b) = x$ , il existe une application  $\phi$  et un élément  $b' \in \text{MEns}(\mathbb{T} \times \mathcal{I})$  tel que  $\pi_1(b') = \pi_1(b)$  et  $\pi_2(b') = x_\phi$ . Plus précisément, nous associons à un jeton  $(d, J_{k'}) \leq b$  un jeton  $(d, I_k)$  tel que  $d \in I_k$ . Réciproquement, étant donné un élément  $b' \in \text{MEns}(\mathbb{T} \times \mathcal{I})$  avec  $\pi_2(b') = x_\phi$ , nous choisissons  $\phi(k, k')$  jetons  $\{(d_i, I_k)\}_{1 \leq i \leq \phi(k, k')}$  et leur substituons les jetons  $\{(d_i, J_{k'})\}_{1 \leq i \leq \phi(k, k')}$ . De cette façon, nous obtenons un multi-ensemble  $b \in \text{MEns}(\mathbb{T} \times \mathcal{I})$  vérifiant  $\pi_2(b) = x$  et  $\pi_1(b) = \pi_1(b')$ .

Le réseau de Petri temporisé avec arcs de lecture résultant de cette transformation est noté  $\mathcal{N}_1$ .  $\square$

**Lemme 5.16.** *Nous pouvons construire un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}_2$ , équivalent à  $\mathcal{N}_1$ , et vérifiant les restrictions (1) et (2).*

**Preuve.** Nous appliquons successivement la transformation suivante à chacune des places de  $\mathcal{N}_1$ . Soit  $p$  une place de  $\mathcal{N}_1$ , et supposons que  $\{[a_1, a_1], ]a_1, a_2[, \dots, ]a_{m-1}, a_m[, [a_m, a_m], ]a_m, a_{m+1}[ \}$  est l'ensemble des intervalles deux à deux disjoints requis par la restriction (1).

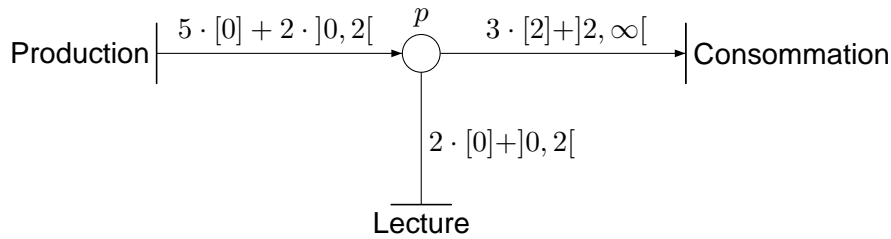
Nous substituons à  $p$  un ensemble de places  $\{p_{a_1}, p_{a_1, a_2}, \dots, p_{a_{m-1}, a_m}, p_{a_m}, p_{a_m, a_{m+1}}\}$ . Nous avons donc besoin après cela de modifier la condition d'acceptation  $\text{Acc}_1$  de  $\mathcal{N}_1$  : la condition d'acceptation  $\text{Acc}_2$  de  $\mathcal{N}_2$  est obtenue en remplaçant toutes les occurrences de  $p$  dans  $\text{Acc}_1$  par le terme  $\sum_{i=1}^m (p_{a_i} + p_{a_i, a_{i+1}})$ . De plus, dans le réseau transformé, un jeton d'âge  $d$  dans la place  $p_{a_i}$  ou  $p_{a_i, a_{i+1}}$  correspondra à un jeton d'âge  $d + a_i$  dans la place  $p$ .

Afin d'utiliser (*i.e.* produire, consommer, ou lire) un jeton d'âge  $a_i$  dans la place  $p$ , il faut donc utiliser un jeton d'âge nul dans la nouvelle place  $p_{a_i}$ . Afin d'utiliser un jeton d'âge  $d \in ]a_i, a_{i+1}[$  dans la place  $p$ , il faut utiliser un jeton d'âge  $d - a_i \in ]0, a_{i+1} - a_i[$  dans la nouvelle place  $p_{a_i, a_{i+1}}$ .

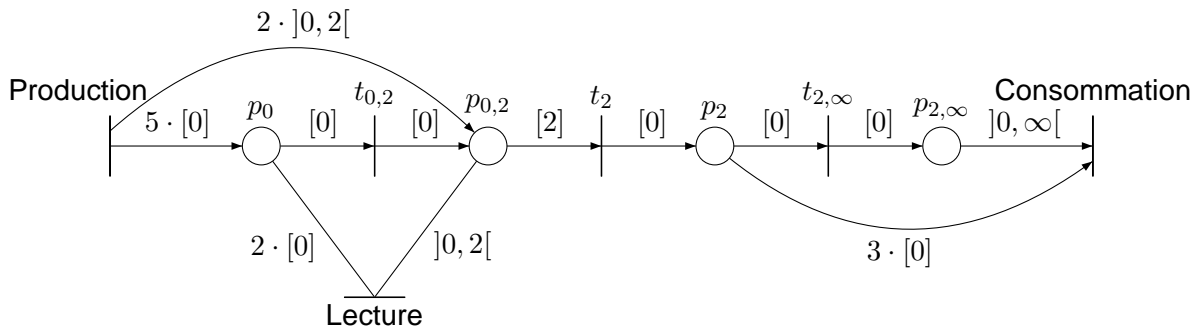
Ensuite, nous transformons un arc connecté à  $p$  avec pour multi-ensemble  $x = n_1 \cdot [a_1, a_1] + n_{1,2} \cdot ]a_1, a_2[ + \dots + n_m \cdot [a_m, a_m] + n_{m, m+1} \cdot ]a_m, a_{m+1}[$  en plusieurs arcs connectés aux nouvelles places tels que le multi-ensemble correspondant à la place  $p_{a_i}$  est  $n_i \cdot [0, 0]$ , et le multi-ensemble correspondant à  $p_{a_i, a_{i+1}}$  est  $n_{i, i+1} \cdot ]0, a_{i+1} - a_i[$ .

Enfin, nous ajoutons des transitions afin de "transférer" les jetons d'une des nouvelles places à une autre lorsque leur âge croît :  $t_{a_1, a_2}, t_{a_2}, \dots, t_{a_m}, t_{a_m, a_{m+1}}$ . Une transition  $t_{a_i}$  consomme un jeton





(a) Exemple de réseau de Petri temporisé.



(b) Réseau obtenu à l'issue de la transformation.

FIG. 5.9 – Illustration de la construction du lemme 5.16.

Dans la séquence précédente, les jetons sont rassemblés par âge identique, par exemple le premier marquage est constitué de sept jetons dans la place  $p$ , cinq d'âge nul, un d'âge 1 et un dernier d'âge 1.2. Cette représentation correspond à la notation que nous avons adoptée pour les multi-ensembles. La séquence de transitions correspondante dans le réseau obtenu est :

$$\text{Production}, (t_{0,2})^5, (0.5), \text{Production}, \text{Production}, \text{Lecture}, (t_{0,2})^{10}, (0.3), t_2, t_{2,\infty}, (0.2), t_2, t_{2,\infty}, (0.5), (t_2)^4, \text{Consommation}$$

┘

**Lemme 5.18.** *Nous pouvons construire un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}_3$ , équivalent à  $\mathcal{N}_2$ , et vérifiant les restrictions (1), (2) et (3).*

**Preuve.** Afin de démontrer ce lemme, nous devons expliquer comment transformer les « sous-réseaux » obtenus dans la preuve du lemme précédent en d'autres réseaux équivalents dans lesquels toutes les places ont la forme d'un des motifs de la figure 5.8.

Nous appliquons successivement les transformations suivantes aux différentes places de  $\mathcal{N}_2$  :

- dupliquer la place pour tout arc de production qui lui est connecté et dupliquer toutes les transitions reliées par un arc de lecture ou un arc de consommation comme cela est représenté sur la figure 5.10(a). Sur cette figure, l'arc connectant la transition  $t$  est représenté en pointillés car il peut s'agir d'un arc de consommation ou d'un arc de lecture. Une telle transition de consommation ou de lecture est donc dupliquée, une copie pour chaque  $0 \leq m \leq n$  si  $n \cdot I$  est le multi-ensemble étiquetant l'arc entre  $p$  et  $t$ .

- dupliquer la place pour tout arc de consommation connecté et dupliquer toutes les transitions connectées par un arc de lecture ou de production à cette place, comme cela est représenté sur la figure 5.10(b). Ici, la transition  $t$  peut être connectée par un arc de production ou un arc de lecture. Une telle transition de production ou de lecture est dupliquée, une copie pour chaque  $0 \leq m \leq n$  si  $n \cdot I$  est le multi-ensemble étiquetant l'arc entre  $p$  et  $t$ .

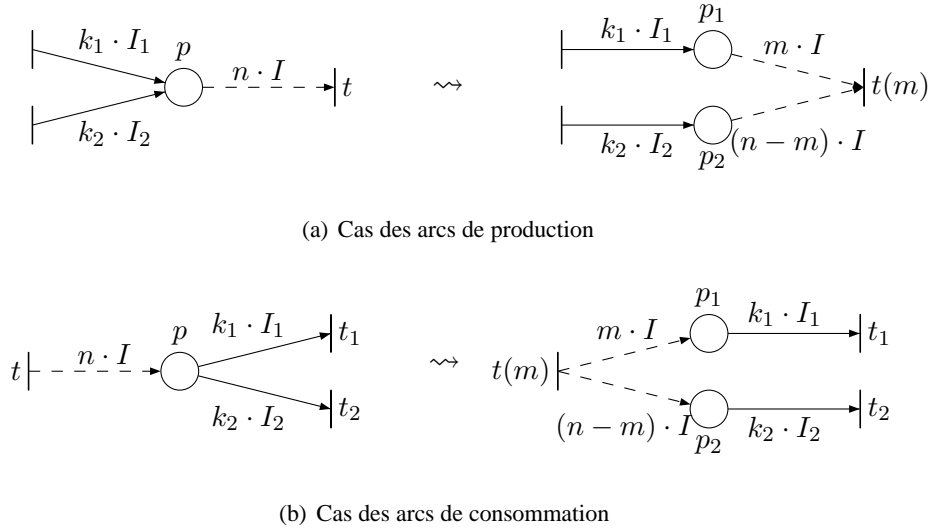


FIG. 5.10 – Duplication des places – constructions du lemme 5.18.

Nous dénotons par  $\mathcal{N}_3$  le nouveau réseau obtenu à l'issue de cette transformation. Nous modifions en conséquence les conditions d'acceptation en remplaçant les occurrences de  $p$  par la somme de ses dupliquées (par exemple  $p_1 + p_2$  si nous avons dupliqué la place  $p$  en deux places  $p_1$  et  $p_2$ ). Il est facile de démontrer que ces constructions conservent le langage accepté par le réseau. Seul un point mérite d'être souligné : dans la dernière transformation, étant donnée une occurrence de  $t$  dans une séquence  $\sigma$  de  $\mathcal{N}_2$ , nous obtenons une séquence correspondante  $\sigma'$  dans  $\mathcal{N}_3$  en choisissant la transition appropriée  $t(m)$  qui dépend de  $\sigma$ . Ainsi, si nous notons  $m_1$  le nombre de jetons produits par  $t$  qui vont être consommés par  $t_1$ , et  $m_2$  le nombre de jetons produits par  $t$  qui vont être consommés par  $t_2$ , un choix correct pour  $m$  devra vérifier l'inégalité  $m_1 \leq m \leq n - m_2$  (remarquons que nous avons nécessairement  $m_1 + m_2 \leq n$ ).

Enfin, les places du réseau  $\mathcal{N}_3$  sont connectées à des arcs de production (respectivement des arcs de consommation) dont les multi-ensembles ont le même intervalle. De plus, grâce à la forme des intervalles obtenus lors de la précédente construction, nous obtenons que toute place est dans l'une des cinq configurations données par les motifs de la figure 5.8.  $\square$

Pour conclure, remarquons que toutes les transformations que nous avons présentées dans cette section préservent à la fois le critère borné des réseaux, ainsi que leur intégralité. Notons également que l'ensemble de la transformation est triplement exponentiel (les entiers étiquetant les arcs sont codés en binaire). Cette borne pourrait sans doute être améliorée, mais ce n'est pas le sujet de notre étude qui ne concerne que l'expressivité des modèles. Ceci conclut donc la preuve de la proposition 5.13.

## 5.5 Expressivité des arcs de lecture

Dans cette section, nous nous intéressons à la question suivante : était-il nécessaire d'ajouter les arcs de lecture au modèle, ou, en d'autres termes, les arcs de lecture ont-ils accru le pouvoir expressif du modèle ?

Grâce au lemme 5.11 (langage  $\mathcal{L}_1$ ), nous savons déjà que l'introduction des arcs de lecture entraîne une augmentation de l'expressivité. En effet, nous avons établi dans ce lemme l'existence d'un langage temporisé reconnu par un réseau avec arcs de lecture, mais qui ne peut pas être reconnu par un réseau sans arcs de lecture. Cependant, le langage considéré reconnaissait uniquement des mots infinis Zeno. Nous allons donc nous intéresser ici à d'autres équivalences de langages, à savoir le cas des mots finis et des mots infinis non Zeno. Nous allons démontrer que pour ces deux équivalences, les arcs de lecture n'accroissent pas le pouvoir d'expression du modèle.

Nous présentons dans la suite de cette section deux constructions différentes : la première, présentée dans la sous-section 5.5.1, est correcte pour l'équivalence de mots finis, la seconde, présentée dans la sous-section 5.5.2, est une adaptation de la première de sorte à traiter l'équivalence de mots infinis non Zeno.

Dans les deux preuves de correction, nous aurons besoin de supposer que les places connectées à des arcs de lecture n'apparaissent pas dans la condition d'acceptation. Cette hypothèse peut être faite sans perte de généralité, comme cela est établi par le lemme suivant :

**Lemme 5.19.** *Étant donné un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}$ , nous pouvons construire effectivement un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}'$  qui est  $\{*, \omega, \omega_{nZ}\}$ -équivalent à  $\mathcal{N}$  et tel qu'aucune place connectée à un arc de lecture n'apparaisse dans la condition d'acceptation.*

**Preuve.** Nous appliquons successivement la transformation suivante à toute place de  $\mathcal{N}$  connectée à un arc de lecture et apparaissant dans une condition d'acceptation. Soit  $p$  une telle place. Le réseau  $\mathcal{N}'$  est obtenu en ajoutant à  $\mathcal{N}$  une nouvelle place  $p'$  telle que pour toute transition  $t \in T$ ,  $t^\bullet(p') = t^\bullet(p)$ ,  ${}^\bullet t(p') = {}^\bullet t(p)$ , et  ${}^\circ t(p') = 0$ . Nous supposons de plus que  $\nu_0(p') = \nu_0(p)$ , et définissons la nouvelle condition d'acceptation de  $\mathcal{N}'$  par celle de  $\mathcal{N}$  dans laquelle la place  $p'$  est substituée à la place  $p$ .

Nous affirmons alors que  $\mathcal{N}'$  est équivalent à  $\mathcal{N}$ . Premièrement, notons qu'étant donnée une configuration accessible de  $\mathcal{N}'$ ,  $p$  et  $p'$  contiennent le même nombre de jetons, mais pas nécessairement des jetons du même âge, puisque les arcs de production peuvent produire des jetons d'âge différents dans un même intervalle, et que les arcs de consommation peuvent choisir de consommer des jetons différents. Cependant, ce dernier point n'est pas grave pour la correction de notre transformation.

Soit  $\sigma'$  une séquence de transitions temporisée de  $\mathcal{N}'$  menant à une configuration acceptante. Alors  $\sigma$ , obtenue à partir de  $\sigma'$  par projection sur les places de  $\mathcal{N}$ , est une séquence de  $\mathcal{N}$ . En effet, comme  $\mathcal{N}$  est un sous-réseau de  $\mathcal{N}'$  obtenu en effaçant des places, tous les comportements de ce dernier sont des comportements du précédent. De plus, grâce à la précédente observation relative aux marquages de  $p$  et  $p'$ , la configuration atteinte à l'issue de  $\sigma$  vérifie la condition d'acceptation de  $\mathcal{N}$ .

Réciproquement, soit  $\sigma$  une séquence de transitions temporisée de  $\mathcal{N}$  menant à une configuration acceptante. Alors nous construisons une séquence  $\sigma'$  de  $\mathcal{N}'$  à partir de  $\sigma$  en consommant et produisant dans la place  $p'$ , des jetons identiques à ceux qui ont été produits et consommés dans la place  $p$  par la séquence  $\sigma$ . La configuration finale de  $\sigma'$  contient les mêmes jetons dans  $p$  et  $p'$  et satisfait donc la condition d'acceptation de  $\mathcal{N}'$ .  $\square$

### 5.5.1 Cas des mots finis

Nous nous intéressons à présent au cas de l'équivalence de mots finis. Avant d'établir notre résultat, nous donnons l'intuition des transformations sur un exemple.

**Exemple 5.20.** Nous considérons le réseau  $\mathcal{N}_1$  représenté sur la figure 5.6(a). Nous transformons ce réseau en un nouveau réseau sans arc de lecture, représenté sur la figure 5.11, qui reconnaît le même langage. Les exécutions acceptantes de ce réseau sont les suivantes. À la date 0, la transition silencieuse  $t_1$  produit un jeton d'âge nul dans la place  $p_1$  et un autre dans la place  $p_2$ . Ensuite, des  $a$  sont produits à l'aide de franchissements de la transition  $t_3$  et pour finir, avant qu'une unité de temps se soit écoulée, la transition silencieuse  $t_2$  est franchie. Celle-ci vide les places  $p_1$  et  $p_2$ . Le tir de cette dernière transition est imposé à l'aide de la condition d'acceptation  $p_1 + p_2 = 0$ .  $\lrcorner$

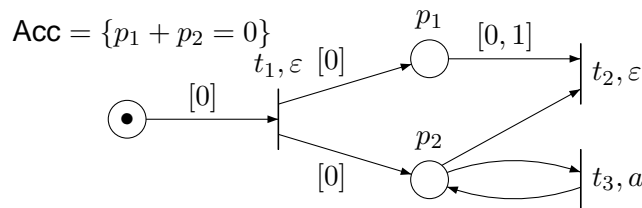


FIG. 5.11 – Une illustration des idées mises en œuvre pour retirer les arcs de lecture.

**Théorème 5.21.** Soit  $\mathcal{N}$  un réseau de Petri temporisé avec arcs de lecture. Nous pouvons construire de façon effective un réseau de Petri temporisé  $\mathcal{N}'$  (sans arcs de lecture), qui est  $*$ -équivalent à  $\mathcal{N}$ . De plus, la transformation préserve les caractères borné et intégral du réseau.

**Preuve.** Afin de démontrer ce résultat, nous procédons dans un premier temps à la normalisation du réseau présentée dans la section 5.4.2. Nous pouvons ensuite distinguer les cinq cas ainsi obtenus représentés sur la figure 5.8 pour une place  $p$ , et montrer que dans chacun de ces cas, nous pouvons effectivement nous passer des arcs de lecture connectés à la place  $p$ .

**Motif  $\mathcal{P}_1$ .** La construction est présentée sur la figure 5.12. Ceci constitue le cas le plus favorable. En effet, la simulation utilisée est la même que celle utilisée dans le cadre non temporisé. Il est facile de vérifier que les séquences de tirs sont exactement les mêmes, et que les deux réseaux sont donc équivalents.

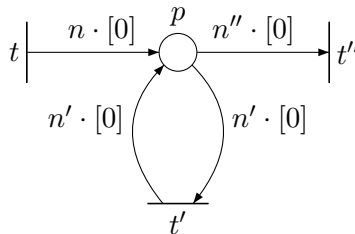


FIG. 5.12 – Retirer les arcs de lecture du motif  $\mathcal{P}_1$



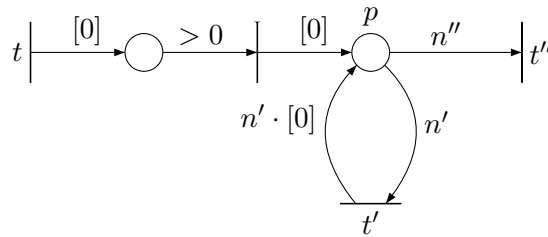


FIG. 5.13 – Retirer les arcs de lecture du motif  $\mathcal{P}_2$ , cas  $a = +\infty$

**Motif  $\mathcal{P}_2$ .** Nous traitons séparément les cas  $a = +\infty$  et  $a < +\infty$ . La construction dans le premier cas est représentée sur la figure 5.13. Pour le second cas, la construction est présentée sur la figure 5.14.

Le cas  $a = +\infty$  est relativement simple. Il suffit en effet de remarquer qu’une fois que le jeton n’est plus d’âge nul, il peut être utilisé indifféremment en lecture et en consommation, son âge ne contraignant plus le franchissement. Notons d’ailleurs que nous n’avons pas modifié la condition d’acceptation.

Le cas  $a < +\infty$  est au contraire plus délicat car nous devons tenir compte de l’âge des jetons. La simulation des arcs de lecture est donc plus difficile. Nous modifions de plus la condition d’acceptation

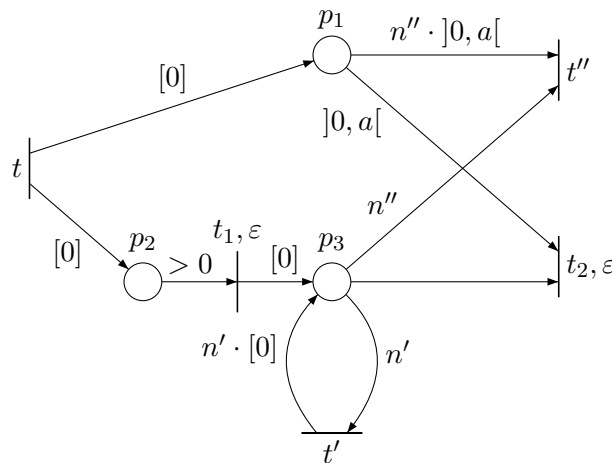


FIG. 5.14 – Retirer les arcs de lecture du motif  $\mathcal{P}_2$ , cas  $a < +\infty$

de  $\mathcal{N}$  en ajoutant la contrainte suivante :  $\sum_{i=1}^3 p_i = 0$ . Avant de démontrer l’équivalence entre les deux réseaux, nous faisons quelques remarques préliminaires sur certains invariants du réseau  $\mathcal{N}'$ . Toute configuration  $\nu$  apparaissant le long d’une séquence de tirs acceptante de  $\mathcal{N}'$  vérifie les propriétés suivantes :

- (i)  $\text{taille}(\nu(p_1)) = \text{taille}(\nu(p_2)) + \text{taille}(\nu(p_3))$
- (ii)  $\text{taille}(\nu(p_2)) \geq \text{taille}(\nu(p_1)|_{=0})$   
où  $\nu(p_1)|_{=0}$  est le multi-ensemble de jetons de la place  $p_1$  dont l’âge est égal à 0
- (iii)  $\text{taille}(\nu(p_1)) = \text{taille}(\nu(p_1)|_{<a})$   
où  $\nu(p_1)|_{<a}$  est le multi-ensemble de jetons de la place  $p_1$  dont l’âge est strictement inférieur à  $a$

Les deux premières propriétés sont de simples invariants obtenus en comparant les arcs de production et de consommation connectés aux places  $p_1$ ,  $p_2$  et  $p_3$ .

La dernière propriété repose sur la condition d'acceptation ajoutée plus haut. En effet, comme la séquence est acceptante, elle doit satisfaire cette condition. Ceci implique que tout jeton produit en place  $p_1$  doit être consommé plus tard par l'une des deux transitions  $t''$  et  $t_2$ . Les contraintes temporelles  $]0, a[$  des arcs connectés à la place  $p_1$  des transitions  $t''$  et  $t_2$  impliquent alors que l'âge de ces jetons ne peut pas atteindre la valeur  $a$ .

Considérons dans un premier temps une séquence de tir acceptante  $\sigma$  de  $\mathcal{N}$ , et construisons une séquence correspondante  $\sigma'$  de  $\mathcal{N}'$ . Nous procédons à deux types de modifications. Premièrement, nous transférons les jetons depuis la place  $p_2$  vers la place  $p_3$  à l'aide la transition silencieuse  $t_1$  dès que nous avons besoin de ces jetons pour le tir d'une transition  $t'$  ou  $t''$  (si un jeton n'est jamais utilisé, nous le déplaçons lorsque son âge est égal à  $\frac{a}{2}$  – rappelons que  $a < +\infty$ ). Deuxièmement, nous vidons les places  $p_1$  et  $p_3$  en utilisant la transition  $t_2$  dès que ces jetons ne sont plus jamais utilisés jusqu'à la fin de la séquence. De cette façon, nous consommons tous les jetons morts de la place  $p$  de  $\mathcal{N}$ . Il est possible de déterminer si un jeton sera encore utilisé ou non car nous ne considérons que des séquences finies (avec notre sémantique, les mots finis ne peuvent être reconnus que par des séquences finies, cf sous-section 1.2.1). Les transitions silencieuses que nous avons insérées permettent de franchir les transitions discrètes de  $\mathcal{N}'$  correspondant à celles de  $\mathcal{N}$  franchies dans  $\sigma$ .

Réciproquement, considérons à présent une séquence de tir  $\sigma'$  de  $\mathcal{N}'$ . nous construisons une séquence  $\sigma$  de  $\mathcal{N}$  obtenue à partir de  $\sigma'$  en effaçant les transitions  $t_1$  et  $t_2$ . Nous vérifions à présent que les transitions  $t'$  et  $t''$  sont encore franchissables dans  $\sigma$ . Tout d'abord, notons que les arcs de production impliquent l'inégalité suivante entre deux configurations  $\nu$  et  $\nu'$  obtenues respectivement après le même préfixe de  $\sigma$  et  $\sigma'$  :

$$\nu(p) \geq \nu'(p_1)$$

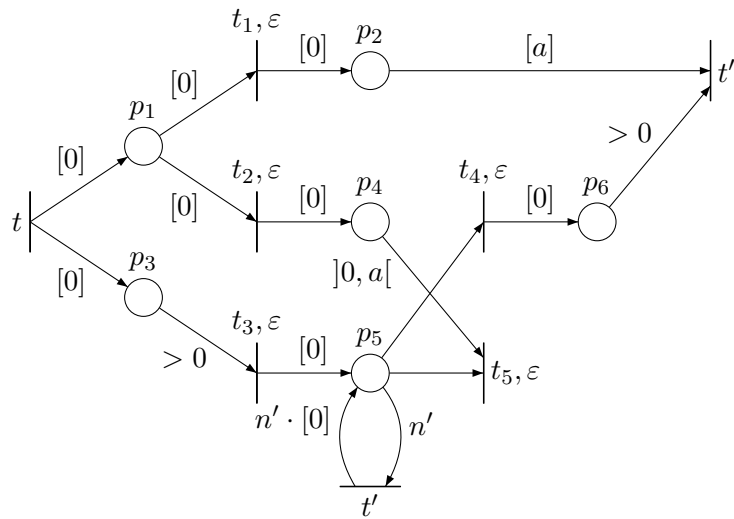
Ceci implique que toute occurrence franchissable d'une transition  $t''$  dans  $\sigma'$  l'est encore dans  $\sigma$ . Afin de démontrer la même propriété pour  $t'$ , nous allons utiliser les propriétés préliminaires démontrées plus haut. Supposons que  $t'$  soit franchissable depuis la configuration  $\nu'$ . Alors la place  $p_3$  contient au moins  $n'$  jetons. Les propriétés (i), (ii) et (iii) énoncées plus haut impliquent alors que la place  $p_1$  contient au moins  $n'$  jetons dont l'âge appartient à l'intervalle  $]0, a[$ . L'inégalité précédente entre  $\nu(p)$  et  $\nu'(p_1)$  donne finalement que la transition  $t'$  est également franchissable depuis  $\nu$  dans  $\mathcal{N}$ . Ceci conclut la preuve de correction pour ce second motif  $\mathcal{P}_2$ .

**Motif  $\mathcal{P}_3$ .** La construction est représentée sur la figure 5.15. Nous modifions à nouveau la condition d'acceptation de  $\mathcal{N}$  en ajoutant la contrainte  $\sum_{i=1}^6 p_i = 0$ . À nouveau, avant de démontrer l'équivalence entre les deux réseaux, nous démontrons quelques invariants du réseau  $\mathcal{N}'$ . Toute configuration  $\nu$  apparaissant le long d'une séquence de tir acceptante de  $\mathcal{N}'$  satisfait les propriétés suivantes :

- (i)  $\text{taille}(\nu(p_1)|_{=0}) + \text{taille}(\nu(p_2)|_{=0}) + \text{taille}(\nu(p_4)|_{=0}) = \text{taille}(\nu(p_3)|_{=0})$
- (ii)  $\text{taille}(\nu(p_2)|_{=a}) \leq \text{taille}(\nu(p_6)|_{>0})$
- (iii)  $\text{taille}(\nu(p_2)|_{>0}) + \text{taille}(\nu(p_4)|_{>0}) = \text{taille}(\nu(p_3)|_{>0}) + \text{taille}(\nu(p_5)) + \text{taille}(\nu(p_6))$
- (iv)  $\text{taille}(\nu(p_2)|_{]0,a[}) + \text{taille}(\nu(p_4)|_{>0}) \geq \text{taille}(\nu(p_3)|_{>0}) + \text{taille}(\nu(p_5))$

La première propriété est un invariant obtenu en comparant les arcs de production et de consommation connectés aux différentes places  $p_1$ ,  $p_2$  et  $p_3$ .

La deuxième propriété repose sur la nouvelle condition d'acceptation. En effet, comme tout jeton d'âge  $a$  dans la place  $p_2$  doit être consommé en temps nul par la transition  $t''$ , cette transition doit être franchissable, et nous obtenons donc l'inégalité (ii).

FIG. 5.15 – Retirer les arcs de lecture du motif  $\mathcal{P}_3$ 

La troisième propriété est obtenue à partir de la première en laissant le temps s'écouler, et en utilisant le fait que la condition d'acceptation implique l'égalité  $\text{taille}(\nu(p_1)_{>0}) = 0$ .

Enfin, la quatrième propriété peut être obtenue à partir des propriétés (ii) et (iii) par soustraction.

Nous considérons d'abord une séquence de tir acceptante  $\sigma$  de  $\mathcal{N}$  et construisons une séquence correspondante  $\sigma'$  dans  $\mathcal{N}'$ .

À chaque fois qu'un jeton est produit par la transition  $t$ , nous déplaçons le jeton correspondant dans la place  $p_1$ . Si ce jeton doit être consommé plus tard par la transition  $t''$ , alors nous utilisons la transition silencieuse  $t_1$  afin de le déplacer dans la place  $p_2$ . Sinon, nous le transférons dans la place  $p_4$  avec la transition  $t_2$ .

De plus, nous transférons également la copie du jeton située dans la place  $p_3$  vers la place  $p_5$  à l'aide de la transition silencieuse  $t_3$  dès que nous en avons besoin pour le tir de la transition  $t'$  (si un jeton n'est jamais testé par  $t'$ , alors nous le transférons lorsque son âge atteint  $\frac{a}{2}$ ). Cet instant doit apparaître après un délai strictement positif puisque l'intervalle de  $t'$  est  $]0, a[$ , ce qui assure que la transition  $t_3$  est franchissable.

Enfin, dès qu'un jeton situé dans la place  $p_5$  n'est plus jamais utilisé jusqu'à la fin de la séquence par la transition  $t'$ , nous franchissons une des deux transitions  $t_4$  et  $t_5$  afin de le consommer. Deux cas sont en fait possibles :

- ou bien le jeton correspondant dans  $\sigma$  est consommé par  $t''$ , et alors nous déplaçons le jeton précédent à l'aide de  $t_4$ . Remarquons que comme le dernier arc de lecture a eu lieu strictement avant que son âge n'atteigne la valeur  $a$ , l'âge du jeton produit dans la place  $p_6$  sera strictement positif lorsque l'âge du jeton correspondant dans la place  $p_2$  aura atteint  $a$ . Ceci implique que la transition  $t''$  sera franchissable.
- ou bien le jeton n'est jamais consommé par  $t''$ , et dans ce cas nous le consommons immédiatement à l'aide de  $t_5$ . Ceci est possible (intervalle  $]0, a[$  sur l'arc relié à  $p_4$ ) puisque la dernière occurrence de  $t'$  apparaît pour un âge de jeton strictement inférieur à  $a$ .

Notons que les modifications précédentes sont possibles si nous avons réalisé les mêmes choix pour les copies des jetons placés dans  $p_1$  et  $p_3$ . De cette façon, nous parvenons à consommer tous les

jetons correspondant à des jetons morts dans la place  $p$  du réseau  $\mathcal{N}$ . Ceci implique que la séquence  $\sigma'$  est acceptante.

Notons également qu'il est possible de déterminer quand un jeton sera ou non encore utilisé car nous considérons des séquences finies.

Enfin, il est possible de démontrer que les transitions silencieuses insérées permettent d'obtenir une séquence franchissable du réseau  $\mathcal{N}'$ .

Réciproquement, considérons une séquence de tir acceptante  $\sigma'$  de  $\mathcal{N}'$ . Nous construisons une nouvelle séquence  $\sigma$  de  $\mathcal{N}$  obtenue à partir de  $\sigma'$  en effaçant les occurrences des transitions  $t_1, t_2, t_3, t_4$  et  $t_5$ . Nous vérifions à présent que les transitions  $t'$  et  $t''$  sont encore franchissables dans  $\sigma$ . D'abord, remarquons que les arcs de production impliquent l'inégalité suivante entre deux configurations  $\nu$  et  $\nu'$  obtenues respectivement à l'issue du même préfixe de  $\sigma$  et  $\sigma'$  :

$$\nu(p) \geq \nu'(p_1) + \nu'(p_2) + \nu'(p_4)$$

En particulier, ceci implique l'inégalité  $\nu(p) \geq \nu'(p_2)$  et donc que toute occurrence franchissable de la transition  $t''$  dans  $\sigma'$  l'est encore dans  $\sigma$ . Afin de démontrer une propriété similaire pour  $t'$ , nous allons utiliser les remarques préliminaires énoncées plus haut. Supposons que  $t'$  soit franchissable depuis  $\nu'$ . Alors la place  $p_5$  contient au moins  $n'$  jetons. Appliquant l'inégalité (iv) et le fait que l'âge de chaque jeton de la place  $p_4$  ne peut jamais dépasser la valeur  $a$  (puisque nous considérons une séquence acceptante), nous obtenons :

$$\text{taille}(\nu(p_2)_{]0, a[}) + \text{taille}(\nu(p_4)_{]0, a[}) \geq n'$$

Ceci implique, en utilisant l'inégalité précédente liant  $\nu$  et  $\nu'$ , que la place  $p$  contient au moins  $n'$  jetons dont l'âge appartient à l'intervalle  $]0, a[$  dans la configuration  $\nu$ . Ceci prouve que  $t'$  est franchissable depuis  $\nu$  et conclut ainsi la preuve de correction pour le motif  $\mathcal{P}_3$ .

**Motif  $\mathcal{P}_4$ .** Nous distinguons à nouveau les deux cas  $a = +\infty$  (figure 5.16) et  $a < +\infty$  (figure 5.17).

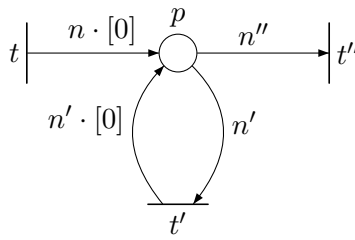


FIG. 5.16 – Retirer les arcs de lecture du motif  $\mathcal{P}_4$ , cas  $a = +\infty$

Pour le cas  $a = +\infty$ , la construction est semblable à celle utilisée pour le motif  $\mathcal{P}_1$ . En effet, un jeton produit est immédiatement utilisable, et l'est à tout jamais grâce à la borne  $a$  infinie. La simulation est donc très simple. Notons à nouveau que nous n'avons pas à forcer certains franchissements avec les conditions d'acceptation.

Comme dans le cas du motif  $\mathcal{P}_2$ , le cas  $a < +\infty$  est au contraire plus délicat car nous devons tenir compte de l'âge des jetons. Nous modifions encore la condition d'acceptation de  $\mathcal{N}$  en ajoutant la contrainte suivante :  $p_1 + p_2 \leq 0$ . La construction utilisée pour ce motif est similaire à celle du motif

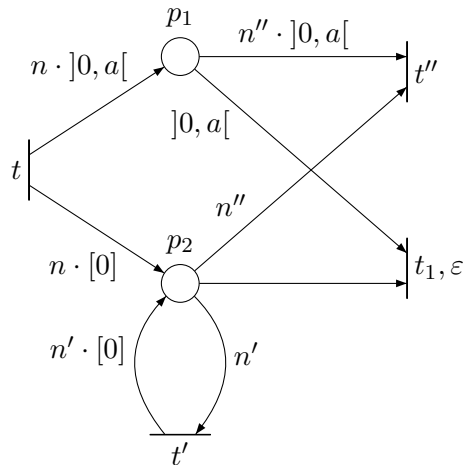


FIG. 5.17 – Retirer les arcs de lecture du motif  $\mathcal{P}_4$ , cas  $a < +\infty$

$\mathcal{P}_2$ . En effet, les arcs de consommation et de lecture sont identiques. La seule différence entre les deux motifs réside donc dans la production des jetons dans la place  $p$ , qui ont ici un âge initial tiré dans l'intervalle  $]0, a[$  (tandis qu'ils étaient d'âge nul dans le motif  $\mathcal{P}_2$ ). Ceci nous permet de proposer une construction plus simple pour ce motif puisque nous n'avons pas à laisser du temps s'écouler avant d'autoriser la transition  $t'$  (correspondant aux arcs de lecture) à lire des jetons produits.

La preuve de correction pour ce motif peut aisément être déduite de celle présentée plus haut pour le motif  $\mathcal{P}_2$ .

**Motif  $\mathcal{P}_5$ .** La construction est présentée sur la figure 5.18. Nous modifions à nouveau la condition

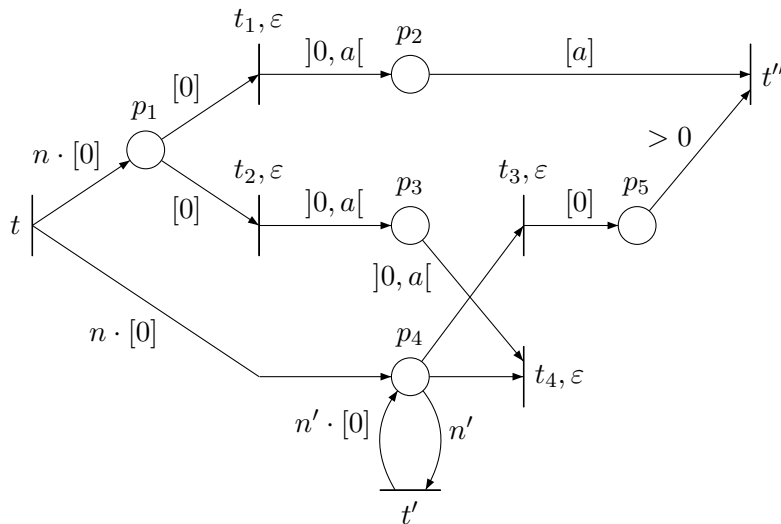


FIG. 5.18 – Retirer les arcs de lecture du motif  $\mathcal{P}_5$

d'acceptation de  $\mathcal{N}$  en ajoutant la contrainte suivante :  $\sum_{i=1}^5 p_i \leq 0$ . Le motif  $\mathcal{P}_5$  est en fait similaire au motif  $\mathcal{P}_3$  puisque les arcs de consommation et de lecture sont les mêmes, et la seule différence provient des arcs de production : alors que les jetons sont produits avec un âge nul dans le motif  $\mathcal{P}_3$ , leur âge est choisi de façon non déterministe dans l'intervalle  $]0, a[$  dans le motif  $\mathcal{P}_5$ .

Nous apportons deux différences notoires par rapport au cas du motif  $\mathcal{P}_2$ .

Remarquons d'abord qu'il n'est pas possible d'imposer que deux jetons produits simultanément dans l'intervalle  $]0, a[$  aient le même âge. Afin d'éviter cette difficulté, nous reportons le choix de l'âge sur les transitions  $t_1$  et  $t_2$ . Rappelons que le choix du tir de  $t_1$  ou de  $t_2$  correspond comme précédemment à la distinction entre les jetons qui seront consommés plus tard par la transition  $t''$  (tirer alors  $t_1$ ) ou qui ne le seront jamais (tirer alors  $t_2$ ).

D'autre part, comme l'âge initial des jetons produits appartient à l'intervalle  $]0, a[$ , ces jetons peuvent immédiatement être utilisés par la transition  $t'$ , et donc, comme pour le motif  $\mathcal{P}_4$ , ceci nous permet de simplifier la construction puisqu'il n'est plus nécessaire de laisser du temps s'écouler avant de transférer les jetons dans la place  $p_4$ .

À nouveau, la preuve de correction pour ce motif peut aisément être déduite de celle présentée plus haut pour le motif  $\mathcal{P}_3$ .

Ceci conclut donc la preuve du théorème 5.21.  $\square$

### 5.5.2 Cas des mots infinis non Zeno

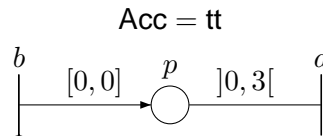


FIG. 5.19 – Un réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}_3$

Dans le cas des mots temporisés infinis, une condition de Büchi similaire imposerait que les places du réseau simulant le réseau initial soient vides infiniment souvent. Cependant, il est possible que ce ne soit pas le cas. Considérons par exemple le réseau  $\mathcal{N}_3$  représenté sur la figure 5.19. Ce réseau accepte le langage de mots temporisés infinis suivant :

$$\mathcal{L}(\mathcal{N}_3) = \{w = (a_i, \tau_i)_{i \geq 0} \mid a_i = a \Rightarrow \exists j < i, a_j = b \text{ et } \tau_i - \tau_j \in ]0, 3[ \}$$

En particulier, le mot temporisé suivant appartient au langage  $\mathcal{L}(\mathcal{N}_3)$  :

$$w = (b, 0)(b, 2)(a, 2)(b, 4)(a, 4) \dots (b, 2i)(a, 2i) \dots$$

Toute configuration de l'unique exécution acceptant  $w$  contient en permanence un jeton dans la place  $p$  qui doit être lu plus tard dans l'exécution et dont la présence est donc nécessaire. Une condition de Büchi similaire à celle utilisée pour les mots temporisés finis éliminerait donc le mot  $w$ . Cependant, dans le cas des mots temporisés pour lesquels le temps diverge, nous pouvons appliquer une transformation au réseau qui ne change pas son langage mais qui permet d'obtenir un réseau pour lequel tout mot temporisé infini non Zeno vérifie une condition de Büchi généralisée appropriée. Intuitivement, ceci consiste pour cet exemple à dupliquer la place  $p$  en deux places et ensuite à produire les jetons alternativement dans l'une ou l'autre des deux copies. Ceci permet de pouvoir « vider » régulièrement chacune des deux copies. Ainsi, chaque copie est vide infiniment souvent (condition de Büchi généralisée), mais les deux copies ne sont pas infiniment souvent vides simultanément (condition de Büchi).

**Théorème 5.22.** *Soit  $\mathcal{N}$  un réseau de Petri temporisé avec arcs de lecture, alors nous pouvons construire de façon effective un réseau de Petri temporisé  $\mathcal{N}'$  (sans arc de lecture), qui est  $\omega_{nZ}$ -équivalent à  $\mathcal{N}$ . De plus, la transformation préserve les caractères borné et intégral du réseau.*

**Preuve.** Nous supposons que le réseau  $\mathcal{N}$  est normalisé, et qu'aucune place reliée à un arc de lecture n'apparaît dans la condition d'acceptation. Remarquons que les seuls motifs ayant des intervalles non bornés sont les motifs  $\mathcal{P}_2$  et  $\mathcal{P}_4$ . Pour ces deux motifs et dans le cas où  $a$  vaut  $+\infty$ , nous n'avons pas modifié la condition d'acceptation dans la transformation présentée pour les mots finis. Nous obtenons donc directement que la simulation démontrée pour les mots finis est également correcte pour ces deux cas pour les mots infinis. Dans la suite, nous ne considérerons donc que le cas où  $a$  est fini.

Nous transformons dans un premier temps  $\mathcal{N}$  en un autre réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}^*$  ainsi. Nous dupliquons toute place  $p$  reliée à un arc de lecture étiqueté par  $]0, a[$  ( $a$  fini), en deux places  $p_{\text{impair}}$  et  $p_{\text{pair}}$ . Nous appliquons ensuite la transformation de façon itérative à chaque place  $p$  et pour chaque arc connecté à  $p$ . Soit  $t$  une transition connectée à  $p$  par un certain arc et notons  $n \cdot I$  le multi-ensemble qui lui est associé. Nous remplaçons  $t$  par un ensemble de transitions  $\{t(k)\}_{0 \leq k \leq n}$  tel que les arcs de ces transitions sont identiques à ceux de  $t$  excepté l'arc considéré. Nous ajoutons à la transition  $t(k)$  deux arcs (de la même nature que celui étudié), un premier étiqueté par  $k \cdot I$  et connecté à  $p_{\text{impair}}$  et un second étiqueté par  $(n - k) \cdot I$  et connecté à  $p_{\text{pair}}$ . Remarquons qu'une transition peut ainsi être dupliquée plusieurs fois. Enfin, l'étiquette des nouvelles transitions est celle de  $t$ .

Il est clair que  $\mathcal{N}$  et  $\mathcal{N}^*$  sont équivalents pour toutes les équivalences de langage et donc en particulier la  $\omega_{nZ}$ -équivalence. Cependant,  $\mathcal{N}^*$  satisfait une propriété supplémentaire que nous allons détailler maintenant. Sélectionnons un entier naturel strictement supérieur à n'importe quelle borne finie apparaissant sur un intervalle du réseau  $\mathcal{N}^*$ . Notons le  $\max$ . Étant donnée une séquence infinie  $\sigma$  et un jeton initialement présent ou produit le long de cette séquence, nous disons qu'un jeton est *inutile* dans une certaine configuration atteinte le long de  $\sigma$  si celui-ci n'est plus jamais utilisé par un arc de lecture ou un arc de consommation (dont l'intervalle est borné) relié à la place qui le contient jusqu'à la fin de  $\sigma$ .

Soit  $w$  un mot temporisé infini non Zeno accepté par une séquence de tir acceptante  $\sigma$  de  $\mathcal{N}$ . Nous construisons alors une séquence  $\sigma^*$  de  $\mathcal{N}^*$  dont l'étiquette est également  $w$  et telle que pour tout  $l \in \mathbb{N}$  :

- il existe une configuration atteinte le long de  $\sigma^*$  à l'instant  $(2l) \cdot \max$  ne contenant que des jetons inutiles dans les places  $p_{\text{pair}}$ ,
- il existe une configuration atteinte le long de  $\sigma^*$  à l'instant  $(2l + 1) \cdot \max$  ne contenant que des jetons inutiles dans les places  $p_{\text{impair}}$ .

La divergence temporelle, due à l'hypothèse que la séquence considérée est non Zeno, intervient maintenant. En effet, elle permet d'affirmer que tout jeton produit dans une certaine place  $p$  ou bien deviendra inutile ou bien sera consommé dans une certaine configuration (caractère borné des intervalles). Si cette configuration appartient à un intervalle de la forme  $[(2l + 1) \cdot \max, (2l + 2) \cdot \max[$ , avec  $l$  un entier naturel, nous dirons que ce jeton est un jeton *impair*. Dans le cas contraire, nous dirons que ce jeton est un jeton *pair*. Nous construisons la séquence  $\sigma^*$  en remplaçant de façon appropriée chaque transition par une de ses dupliquées. Le choix de celle-ci dépend des jetons à consommer, lire ou produire, selon qu'ils sont pairs ou impairs. Ainsi, les jetons pairs (respectivement impairs) sont produits dans la place  $p_{\text{pair}}$  (respectivement la place  $p_{\text{impair}}$ ).

Considérons à présent la dernière configuration de  $\sigma^*$  atteinte à l'instant  $(2k + 1) \cdot \max$  et supposons que la place  $p_{\text{impair}}$  contienne un jeton qui ne soit pas encore inutile. Mais alors celui-ci va devenir inutile au cours de l'intervalle  $](2k + 1) \cdot \max, (2k + 2) \cdot \max[$ , car nous avons supposé  $a$  fini. Ce jeton

est donc un jeton pair, et il aurait dû être produit dans la place  $p_{\text{pair}}$ , ce qui fournit une contradiction. La preuve est similaire pour la dernière configuration atteinte par  $\sigma^*$  à l'instant  $(2k) \cdot \max$ .

Nous appliquons alors la transformation du Théorème 5.21 au réseau  $\mathcal{N}^*$ , obtenant ainsi le nouveau réseau  $\mathcal{N}'$ . Dans la transformation des motifs 2, 3, 4 et 5 lorsque  $a$  est fini, nous enregistrons le caractère *pair* ou *impair* des nouvelles places. Par exemple, dans le cas du motif  $\mathcal{P}_4$ , une place  $p_{\text{impair}}$  sera remplacée par deux places  $p_{\text{impair},1}$  et  $p_{\text{impair},2}$ . Nous modifions alors la condition d'acceptation du réseau en lui ajoutant une condition de Büchi généralisée. Celle-ci impose qu'infiniment souvent, la somme des jetons présents dans les places paires (respectivement impaires) est nulle.

Soit  $w$  un mot temporisé infini non Zeno accepté par  $\mathcal{N}$  (et donc aussi par  $\mathcal{N}^*$ ). Considérons également une séquence  $\sigma^*$  de  $\mathcal{N}^*$  acceptant  $w$  vérifiant la propriété supplémentaire décrite plus haut. Simulons cette séquence dans  $\mathcal{N}'$  afin d'obtenir une nouvelle séquence  $\sigma'$  comme cela a été fait dans la preuve du Théorème 5.21 excepté que les jetons non consommés par  $\sigma^*$  sont consommés dans  $\mathcal{N}'$  dès qu'ils deviennent inutiles à l'aide des transitions silencieuses qui permettent de vider les places. Grâce à la propriété énoncée plus haut pour  $\sigma^*$ , cette nouvelle séquence  $\sigma'$  vérifie les nouvelles conditions exprimées par la condition de Büchi généralisée.

Réciproquement, considérons une séquence infinie non Zeno  $\sigma'$  de  $\mathcal{N}'$  et supposons qu'elle triche. Alors certains jetons des places paires ou impaires ne seront jamais consommés dans  $\sigma'$  et  $\sigma'$  ne peut donc pas être acceptante. Ainsi, pour une séquence acceptante  $\sigma'$  de  $\mathcal{N}'$ , appliquer exactement les mêmes transformations que celles présentées dans le Théorème 5.21 permet d'obtenir une séquence acceptante  $\sigma^*$  de  $\mathcal{N}^*$ . Ceci conclut donc la preuve du Théorème 5.22.  $\square$

**Exemple 5.23** (Application de la construction du théorème 5.22). Considérons le réseau  $\mathcal{N}_3$  représenté sur la Figure 5.19. Il est facile de vérifier que le réseau représenté sur la figure 5.20, disons  $\mathcal{N}'_3$ , est le réseau obtenu par la construction présentée dans la preuve du théorème 5.22. En effet, l'unique place  $p$  de  $\mathcal{N}_3$  est configurée selon le motif  $\mathcal{P}_2$ . La construction consiste donc à dupliquer cette place en deux copies appelées « pair » et « impair » puis à appliquer la construction décrite pour les mots finis à chacune de ces copies. La condition d'acceptation est une condition de Büchi généralisée imposant que les deux ensembles de places obtenus pour chaque copie soient chacun vides infiniment souvent. Nous utilisons l'indice 0 pour représenter les éléments de la copie paire et l'indice 1 pour la copie impaire. Rappelons que le mot suivant est accepté par  $\mathcal{N}_3$ .

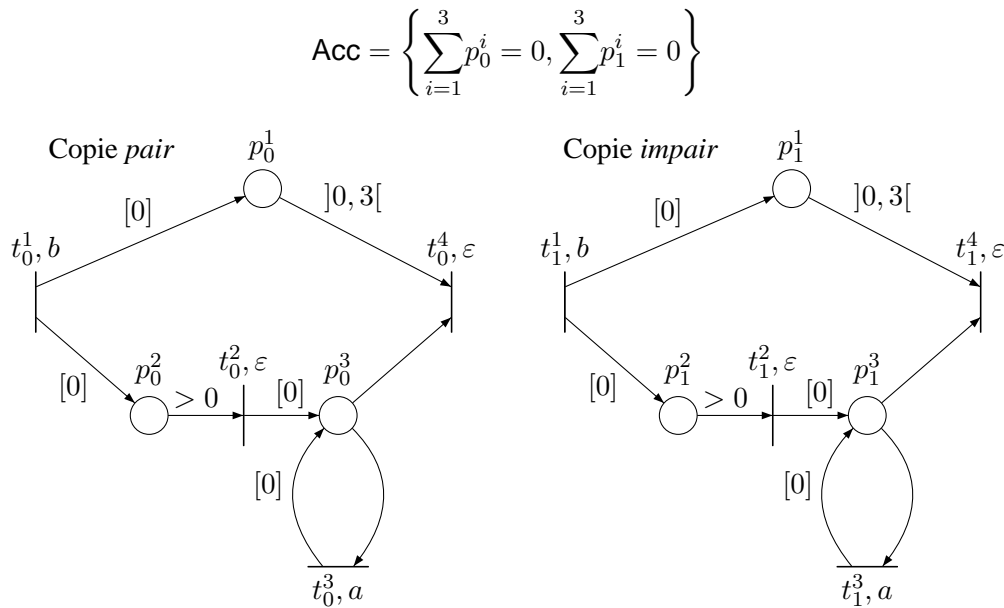
$$w = \begin{array}{cccccccc} (b, 0) & (b, 2) & (a, 2) & (b, 4) & (a, 4) & \dots & (b, 2i) & (a, 2i) & \dots \\ \text{impair} & \text{pair} & & \text{pair} & & & & & \end{array}$$

Nous donnons ici l'exécution correspondante  $\sigma'$  de  $\mathcal{N}'_3$ , comme elle est définie dans la preuve du théorème 5.22. Notons que nous aurions pu exhiber une exécution plus simple pour cet exemple précis. Par définition, nous considérons la valeur 4 pour la constante  $\max$  de la preuve. Un jeton est alors dit pair si il devient inutile dans un intervalle de la forme  $[(2k+1).4, (2k+2).4[$ , et impair sinon. Nous avons indiqué sous les occurrences de  $b$  dans  $w$  si le jeton produit est un jeton pair ou un jeton impair. D'après ces indications, nous pouvons construire la séquence  $\sigma'$  suivante :

$$\sigma' = (t_1^1, 0)(t_1^2, 2)(t_0^1, 2)(t_1^3, 2)(t_1^4, 2)(t_0^1, 4)(t_0^2, 4)(t_0^3, 4)(t_0^4, 4)(t_1^1, 6)(t_0^2, 6)(t_0^3, 6)(t_0^4, 6) \dots$$

Notons  $\nu_1$  (respectivement  $\nu_2$ ) la configuration atteinte après avoir franchi les 5 premières transitions (respectivement 13) de  $\sigma'$ . Il est facile de vérifier que la configuration  $\nu_1$  satisfait la condition d'acceptation  $\sum_{i=1}^3 p_1^i = 0$  et que  $\nu_2$  satisfait la condition d'acceptation  $\sum_{i=1}^3 p_0^i = 0$ .  $\lrcorner$




 FIG. 5.20 – Application du théorème 5.22 au réseau  $\mathcal{N}_3$ 

## 5.6 Expressivité des mises à jour non déterministes

Dans cette section, nous étudions le rôle des mises à jour non déterministes dans les réseaux de Petri temporisés avec arcs de lecture.

Grâce au Lemme 5.12 (langage temporisé  $\mathcal{L}_2$ ), nous savons que la classe des réseaux de Petri temporisés avec arcs de lecture intégraux est strictement plus expressive que la classe des réseaux de Petri temporisés avec arcs de lecture intégraux à remises à zéro. Nous allons prouver ici deux résultats, qui montrent que c'est en fait la combinaison de la présence des arcs de lecture avec le caractère intégral qui est à l'origine du saut d'expressivité entre les réseaux à remises à zéro et les réseaux à mises à jour générales. Nous présentons pour cela deux constructions permettant de s'affranchir des mises à jour générales.

Dans un premier temps, nous proposons dans la sous-section 5.6.1 une construction simple et correcte pour les réseaux sans arcs de lecture qui préserve le caractère intégral des réseaux. Nous présentons ensuite dans la sous-section 5.6.2 une seconde construction nettement plus complexe, valable pour tout réseau, mais qui ne préserve pas le caractère intégral (elle nécessite de raffiner la granularité du réseau).

### 5.6.1 En absence d'arc de lecture

Nous proposons donc dans un premier temps une construction simple permettant de traiter le cadre des réseaux de Petri temporisés sans arc de lecture.

**Théorème 5.24.** *Pour tout réseau de Petri temporisé sans arc de lecture  $\mathcal{N}$ , nous pouvons construire de façon effective un réseau de Petri sans arc de lecture à remises à zéro  $\mathcal{N}'$  qui est  $\{*, \omega, \omega_{nz}\}$ -équivalent à  $\mathcal{N}$ . De plus, cette transformation préserve les caractères borné et intégral du réseau.*

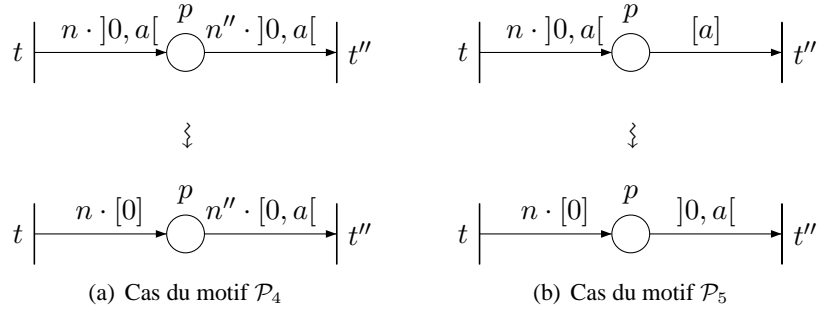


FIG. 5.21 – Retirer les mises à jour non déterministes dans les réseaux de Petri temporisés sans arc de lecture.

Ce résultat ne présente pas de difficulté majeure et consiste simplement en un décalage des intervalles.

**Preuve.** Soit  $\mathcal{N}$  un réseau de Petri temporisé sans arc de lecture. En remarquant que les transformations utilisées dans la preuve de la proposition 5.13 préservent l'absence d'arcs de lecture, nous pouvons supposer que toute place  $p$  de  $\mathcal{N}$  est configurée selon l'un des cinq motifs représentés sur la figure 5.8, dans lesquels les arcs de lecture sont omis.

Seuls les motifs  $\mathcal{P}_4$  et  $\mathcal{P}_5$  contiennent des mises à jour autres que des remises à zéro, nous ne devons donc décrire une transformation que pour ces deux cas. Les constructions correspondantes sont représentées sur la figure 5.21, et leur correction est assez facile à obtenir. Nous en esquissons la preuve dans le cas du motif  $\mathcal{P}_4$ . Si, dans le réseau initial, un jeton est produit dans la place  $p$  avec un âge initial  $x \in ]0, a[$  et est consommé lorsque son âge vaut  $y \in ]0, a[$ , alors dans le nouveau réseau obtenu *via* notre transformation, il est possible de produire un jeton dans  $p$  d'âge nul et de le consommer lorsque son âge vaut  $y - x \in [0, a[$ . Réciproquement, si un jeton est produit dans  $p$  (avec un âge nul) dans le nouveau réseau et quitte la place  $p$  avec un âge  $x \in [0, a[$ , alors il est possible dans le réseau original de produire dans  $p$  un jeton d'âge  $\frac{a-x}{2} \in ]0, a[$  si  $a < \infty$  (et d'âge 1 sinon), puis de le consommer lorsque son âge vaut  $\frac{a+x}{2} \in ]0, a[$  si  $a < \infty$  (ou  $1 + x$  sinon). Ainsi, les jetons morts du premier réseau correspondent aux jetons morts du second. Ceci conclut le cas du motif  $\mathcal{P}_4$  et le motif  $\mathcal{P}_5$  peut être traité de façon similaire.  $\square$

## 5.6.2 En présence d'arcs de lecture

La seconde construction que nous présentons dans cette sous-section est nettement plus délicate et requiert de raffiner la granularité du réseau, mais permet de traiter l'ensemble de la classe des réseaux de Petri temporisés avec arcs de lecture.

**Théorème 5.25.** *Pour tout réseau de Petri temporisé avec arcs de lecture  $\mathcal{N}$ , nous pouvons construire de façon effective un réseau de Petri temporisé avec arcs de lecture à remises à zéro  $\mathcal{N}'$  qui est  $\{*, \omega_{nz}, \omega\}$ -équivalent à  $\mathcal{N}$ . Cette transformation préserve le caractère borné du réseau, mais **pas** son éventuel caractère intégral.*

**Preuve.** Tout d'abord, il est important de remarquer que dans le cas des mots finis et infinis non Zeno, ce résultat est un corollaire de résultats précédents (théorèmes 5.21, 5.22 et 5.24). La construction que nous présentons maintenant, bien que correcte également dans le cadre des équivalences de mots finis et infinis non Zeno, n'est donc strictement nécessaire que pour traiter l'équivalence de mots infinis.

Soit  $\mathcal{N}$  un réseau de Petri temporisé avec arcs de lecture dont nous supposons qu'il vérifie les conclusions de la proposition 5.13. Les seules places de  $\mathcal{N}$  qui sont reliées à des arcs de production ne correspondant pas à des remises à zéro sont donc celles qui correspondent aux motifs  $\mathcal{P}_4$  et  $\mathcal{P}_5$  (figures 5.8(d) et 5.8(e)). Nous allons donc présenter deux transformations, une pour chacun de ces deux motifs.

**Cas du motif  $\mathcal{P}_4$ .** La construction pour ce cas est représentée sur la figure 5.22. Nous notons  $\mathcal{N}'$  le réseau de Petri temporisé résultant de la transformation. Nous démontrons à présent l'équivalence entre les deux réseaux  $\mathcal{N}$  et  $\mathcal{N}'$ .

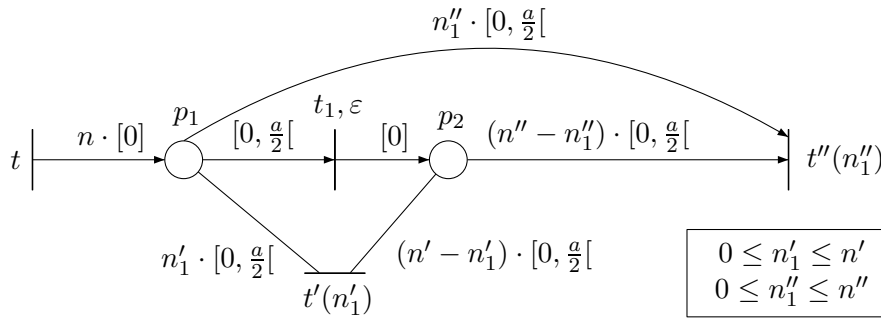


FIG. 5.22 – Équivalent à remises à zéro du motif  $\mathcal{P}_4$

Soit  $\sigma$  une séquence de tir infinie acceptante de  $\mathcal{N}$ . Nous construisons une nouvelle séquence  $\sigma'$  dans  $\mathcal{N}'$  acceptant le même mot temporisé comme suit.

Choisissons un jeton de la place  $p$  d'âge initial  $\delta$ . Deux cas peuvent alors se produire :

- *Premier cas* : ce jeton ne sera jamais consommé par  $t''$ . Si  $\delta \geq \frac{a}{2}$ , alors nous le laissons définitivement dans  $p_1$ . Sinon ( $0 < \delta < \frac{a}{2}$ ), après avoir laissé s'écouler  $\frac{a}{2} - \delta$  unités de temps, nous le transférons dans  $p_2$  à l'aide de la transition silencieuse  $t_1$ . Remarquons que le jeton dans  $\mathcal{N}'$  est donc disponible au moins aussi longtemps dans  $p_1$  ou  $p_2$  que le jeton correspondant l'est dans  $\mathcal{N}$ .
- *Second cas* : ce jeton est consommé par  $t''$  lorsque son âge vaut  $\delta'$ . Si  $0 < \delta' - \delta < a$ , alors nous le transférons dans  $p_2$  après avoir laissé s'écouler  $\frac{\delta' - \delta}{2}$  unités de temps. Sinon, le jeton est immédiatement consommé et le temps ne s'écoule pas : nous n'avons donc pas besoin de transférer le jeton. Remarquons à nouveau que le jeton de  $\mathcal{N}'$  est donc disponible au moins aussi longtemps dans  $p_1$  ou  $p_2$  que celui de  $\mathcal{N}$  l'est.

À présent, la séquence  $\sigma'$  est obtenue à partir de  $\sigma$  en insérant des occurrences de transitions de transfert et en substituant à  $t'$  (respectivement  $t''$ ) la transition appropriée  $t'(n_1')$  (respectivement  $t''(n_1'')$ ) selon les places dans lesquelles se trouvent les jetons correspondant (dans  $\mathcal{N}'$ ) à ceux de  $p$  qui sont utilisés par le tir de  $t'$  (respectivement  $t''$ ) dans  $\mathcal{N}$ .

Réciproquement, soit  $\sigma'$  une séquence de tir infinie acceptante dans  $\mathcal{N}'$ . Nous construisons une nouvelle séquence  $\sigma$  dans  $\mathcal{N}$  acceptant le même mot temporisé comme suit.

Nous effaçons simplement les occurrences des transitions de transfert et nous substituons aux transitions  $t'(n_1')$  (respectivement  $t''(n_1'')$ ) la transition  $t'$  (respectivement  $t''$ ). Il reste à définir l'âge initial d'un jeton produit dans  $p$ . Si ce jeton correspond à un jeton de  $\mathcal{N}'$  qui n'est pas transféré dans  $p_2$ , alors nous lui affectons l'âge initial  $\frac{a}{2}$ . Si le jeton de  $\mathcal{N}'$  est transféré dans  $p_2$  lorsque son âge vaut  $\delta$ , alors nous affectons au jeton de  $\mathcal{N}$  l'âge initial  $\frac{a}{2} - \delta$ . Grâce à ce choix, le jeton de  $\mathcal{N}$  est disponible

au moins aussi longtemps que celui de  $\mathcal{N}'$  l'est dans  $p_1$  ou  $p_2$  et ainsi toute transition franchissable dans  $\sigma'$  l'est également dans  $\sigma$ .

Ceci conclut la preuve dans le cas du motif  $\mathcal{P}_4$ .

**Cas du motif  $\mathcal{P}_5$ .** Cette construction est plus délicate car les actions de lecture et les consommations ont lieu dans des intervalles différents ( $]0, a[$  et  $[a$  respectivement). Afin de mieux comprendre les problèmes supplémentaires présents dans le motif  $\mathcal{P}_5$ , nous commençons par proposer une simulation erronée (représentée sur la figure 5.23) directement adaptée de la simulation précédente.

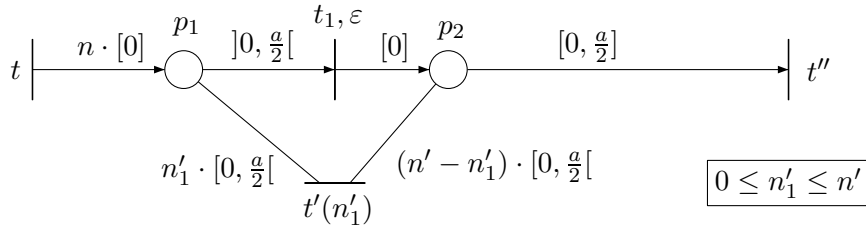


FIG. 5.23 – Une simulation à remise à zéro erronée pour le motif  $\mathcal{P}_5$

En utilisant une preuve similaire à celle développée pour le motif  $\mathcal{P}_4$ , il est possible de démontrer que toute séquence de tir  $\sigma$  de  $\mathcal{N}$  peut être simulée dans ce réseau. Cependant, le contraire est faux : ce réseau permet de produire des séquences qui n'existent pas dans  $\mathcal{N}$ . Supposons par exemple que  $n = n' = 1$ , alors la séquence de tir  $(t, 0)(t_1, \frac{a}{4})(t'(1), \frac{a}{4})(t'', \frac{a}{4})$  ne correspond à aucune séquence du réseau original. En effet, si une telle séquence existait, alors l'âge du jeton produit par la transition  $t$  appartiendrait à l'intervalle  $]0, a[$  à l'instant  $\frac{a}{4}$  afin de pouvoir franchir  $t'$ . Mais au même instant, la transition  $t''$  ne peut alors pas être franchissable puisque l'âge de ce jeton n'appartient pas à l'intervalle  $[a$ . Le problème de cette simulation est qu'à un même instant, un jeton peut être utilisé pour d'abord simuler un franchissement de  $t'$  et ensuite pour simuler un franchissement de  $t''$ .

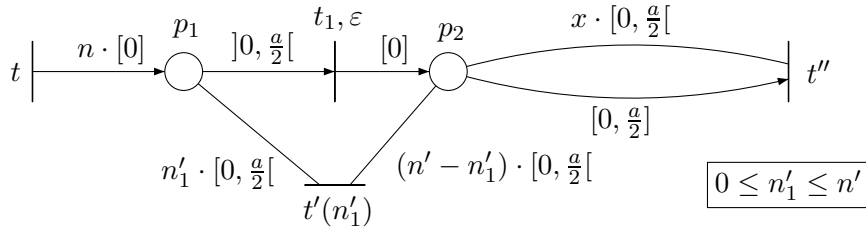


FIG. 5.24 – Une simulation à remise à zéro du motif  $\mathcal{P}_5$  ... avec un poids dynamique

Nous présentons maintenant une seconde simulation (représentée sur la figure 5.24) qui est correcte mais utilise un poids « dynamique »  $x$  pour un de ses arcs. Expliquons la sémantique de celui-ci : lorsque  $t''$  est franchie à un certain instant  $\tau$ ,  $x$  représente le maximum des valeurs  $n' - n'1$  telles qu'un franchissement  $t'(n'1)$  a eu lieu à la même date  $\tau$ . Ainsi, le problème rencontré par la précédente simulation est résolu. Cependant, notre modèle ne contient pas de tels poids dynamiques et la transformation correcte représentée sur la figure 5.25 consiste essentiellement à simuler un tel poids dynamique. Nous notons à nouveau  $\mathcal{N}'$  le réseau de Petri temporisé avec arcs de lecture obtenu par cette transformation.

Avant de démontrer la correction de cette construction, nous donnons quelques explications sur  $\mathcal{N}'$ . Premièrement, la place *actif* est reliée à toute transition de  $\mathcal{N}$  par un arc de lecture dont l'étiquette

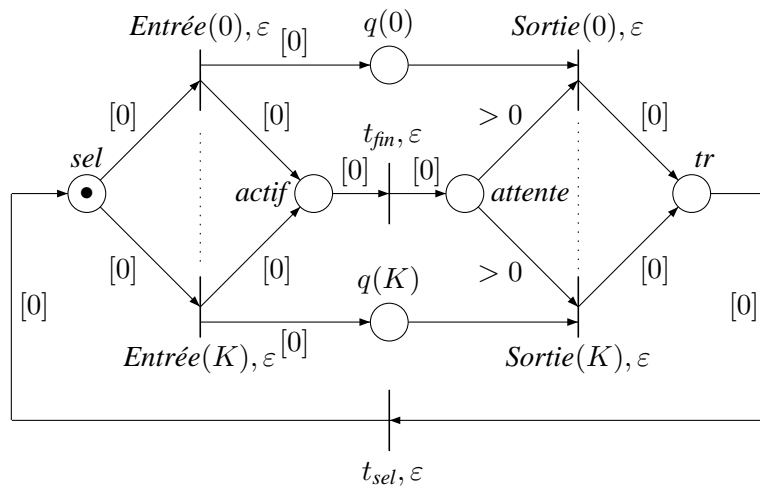
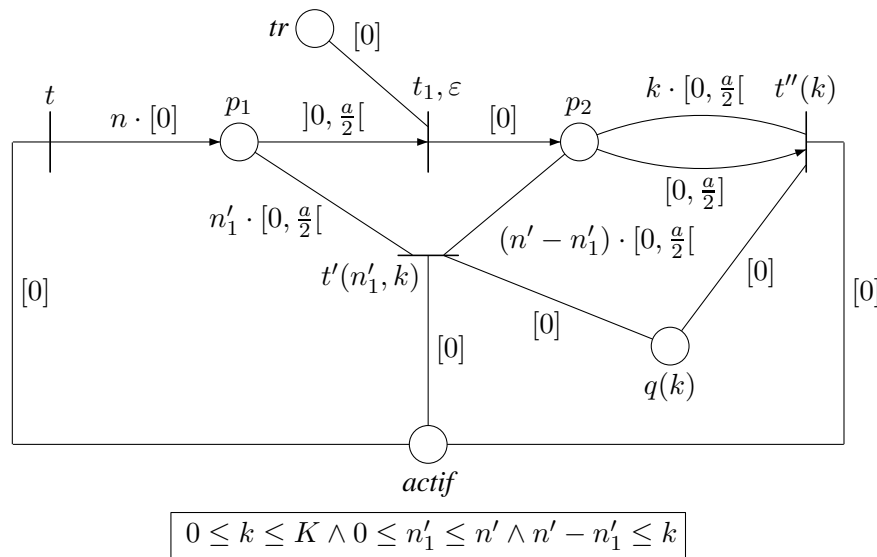


FIG. 5.25 – Équivalent à remises à zéro pour le motif  $\mathcal{P}_5$ .

vaut  $[0]$ . Deuxièmement, en notant  $K$  la plus grande constante  $n'$  apparaissant sur une étiquette  $n' \cdot ]0, a[$  d'un arc de lecture, nous définissons, pour tout indice  $k$  tel que  $0 \leq k \leq K$ , la place  $q(k)$  et deux transitions  $Entrée(k)$  et  $Sortie(k)$ .

La partie inférieure du réseau joue trois rôles. D'abord, elle permet d'ordonner la partie haute du réseau : elle rend en effet explicite l'alternance entre les écoulements de temps et les séquences de tir instantanées de la partie haute du réseau utilisées pour simuler les séquences de tir instantanées du réseau original. De plus, avant de simuler de telles séquences instantanées, elle permet de sélectionner le nombre maximal de jetons qui vont être testés par des tirs de  $t'$  au cours de cette séquence (ce nombre correspond à la valeur  $k$  du poids dynamique de la simulation précédente). Enfin, après l'écoulement du temps, elle déplace les jetons de  $p_1$  à  $p_2$  afin d'assurer que ces transferts n'ont pas lieu au cours de la simulation d'une séquence instantanée. Plus précisément, tout comportement du réseau  $\mathcal{N}'$  doit être une itération (éventuellement infinie) de la séquence décrite ci-après. Initialement,

la place *sel* contient un jeton d'âge nul.

- La première étape est la « sélection » (ce qui justifie le nom de la place *sel*) d'un indice  $k$  parmi l'ensemble  $\{0, \dots, K\}$  (voir ci-dessus). Cette sélection est réalisée par la consommation par une unique transition  $Entrée(k)$  appartenant à la famille  $(Entrée(k))_{0 \leq k \leq K}$  du jeton situé dans *sel*. Ce tir place ainsi instantanément (*i.e.* après un temps nul) un jeton dans la place  $q(k)$  correspondante ainsi que dans la place *actif*.
- Une fois que la place *actif* contient un jeton, une phase de tir instantanée des transitions de  $\mathcal{N}$  est « activée » (d'où le nom de la place *actif*). En effet, le réseau peut tirer instantanément les transitions correspondant à celles de  $\mathcal{N}$ , *i.e.* les transitions correspondant à  $t$ ,  $t'$  et  $t''$ . Ensuite, encore en temps nul, la transition  $t_{fin}$  est franchie, signifiant la « fin » de cette phase de tir, et le jeton de la place *actif* est consommé et un nouveau jeton est produit dans la place *attente*.
- Une transition de délai intervient alors nécessairement de sorte que le jeton de la place *attente* ait un âge strictement positif, rendant ainsi possible le tir de la transition silencieuse  $Sortie(k)$ . Celle-ci consomme le jeton de la place  $q(k)$  et en produit un dans la place *tr*.
- La partie supérieure du réseau peut alors transférer (grâce au jeton situé dans la place *tr*) en temps nul certains jetons de la place  $p_1$  vers la place  $p_2$ , à l'aide de la transition  $t_1$ .
- Enfin, la transition silencieuse  $t_{sel}$  est tirée en temps nul et reproduit ainsi un jeton dans la place *sel* de la partie inférieure.

Nous pouvons à présent démontrer que les deux réseaux sont équivalents.

Dans un premier temps, considérons une séquence de tir infinie acceptante  $\sigma$  dans  $\mathcal{N}$ . Nous construisons à partir de celle-ci une nouvelle séquence  $\sigma'$  dans  $\mathcal{N}'$  acceptant le même mot temporisé. Pour cela, nous commençons par ajouter à  $\sigma$  des informations supplémentaires et nous supposons de plus que le choix des marquages intermédiaires atteints le long de  $\sigma$  est fixé et que les jetons sont individualisés.

Décrivons d'abord comment attacher une « date de transfert » (éventuellement infinie) à chaque jeton de  $p$ . Remarquons que la technique est similaire à celle utilisée pour le motif  $\mathcal{P}_4$ . Choisissons donc un jeton de  $p$  d'âge initial  $\delta$  produit à la date  $\tau$ . Deux cas peuvent se produire :

- *Premier cas* : ce jeton ne sera jamais consommé par  $t''$ . Si  $\delta \geq \frac{a}{2}$ , alors nous affectons  $+\infty$  à sa date de transfert. Sinon ( $0 < \delta < \frac{a}{2}$ ), sa date de transfert vaut  $\tau + (\frac{a}{2} - \delta)$ .
- *Second cas* : ce jeton est consommé par  $t''$  (nécessairement lorsque son âge vaut alors  $a$ ). Sa date de transfert vaut alors  $\tau + \frac{a-\delta}{2}$ .

Considérons à présent une séquence de tir instantanée maximale  $\rho$ , *i.e.* une sous-séquence maximale (éventuellement infinie) de  $\sigma$  dont la longueur temporelle est nulle. Dans cette sous-séquence, nous associons à toute occurrence d'une transition  $t'$  connectée à  $p$  par un arc de lecture d'étiquette  $n' \cdot ]0, a[$  le nombre de jetons testés par cet arc qui n'ont pas encore atteint leur date de transfert. Notons ce nombre  $n'_1$ . Nous affectons alors à l'ensemble de la sous-séquence  $\rho$  la valeur maximale (finie) parmi les valeurs  $n' - n'_1$  pour de tels  $n'_1$  (0 si cet ensemble est vide). Nous notons cet entier naturel  $k$  :  $k = \max\{n' - n'_1 \mid n'_1 \text{ est associé à } t', t' \in \rho\}$ . Nous avons en particulier  $k \leq K$ .

Nous construisons à présent la séquence  $\sigma'$  comme suit. Soit  $0 = \tau_0 < \tau_1 < \tau_2 < \dots$  la séquence strictement croissante (finie ou infinie) des dates correspondant à des dates de franchissement de transitions dans  $\sigma$  ou à des dates de transfert (ou aux deux).

Dans  $\sigma'$ , la « partie basse » du réseau de la figure 5.25 décompose les transitions de délai selon la partition précédente et selon le comportement général de cette partie du réseau qui a été décrit précédemment. Décrivons plus précisément comment nous simulons  $\sigma$  dans  $\sigma'$ . Supposons d'abord que  $\tau_i$  corresponde à une date de franchissement dans  $\sigma$  et notons  $\rho$  la sous-séquence instantanée maximale franchie à la date  $\rho$ . Alors dans  $\sigma'$  nous sélectionnons la valeur  $k$  associée à  $\rho$  décrite

plus haut en franchissant la transition  $Entrée(k)$ . Ensuite, la partie haute du réseau peut simuler  $\rho$  en substituant à toute occurrence de  $t'$  (respectivement de  $t''$ ) la transition  $t'(n'_1, k)$  (respectivement  $t''(k)$ ) où  $n'_1$  a été défini plus haut. Ensuite, après avoir franchi la transition  $t_{fin}$ , nous laissons  $\tau_{i+1} - \tau_i$  unités de temps s'écouler, franchissons  $Sortie(k)$  et franchissons enfin la transition  $t_1$  autant de fois que nécessaire de sorte à transférer tous les jetons dont la date de transfert vaut  $\tau_{i+1}$ . Finalement, nous ajoutons à la séquence  $\sigma'$  une occurrence de la transition  $t_{sel}$ .

Nous affirmons que nous obtenons de cette façon une séquence de tir dans  $\mathcal{N}'$  acceptant le même mot temporisé. La correction des franchissements des transitions  $t'(n'_1, k)$  s'obtient d'une façon similaire à la preuve du motif  $\mathcal{P}_4$ . Le seul point nécessitant d'être détaillé est la validité du tir de  $t''(k)$  dans  $\mathcal{N}'$  puisqu'il y a un arc de lecture supplémentaire. Cependant, ce franchissement intervient dans une sous-séquence instantanée maximale dans laquelle  $k$  jetons ont été lus dans  $p_2$  avec un âge appartenant à l'intervalle  $[0, a/2[$ . Grâce à nos choix de franchissements de la transition de transfert  $t_1$ , ces jetons correspondent dans  $\mathcal{N}$  à des jetons de la place  $p$  dont l'âge est strictement inférieur à  $a$  au cours de cette sous-séquence. Ainsi, ils ne peuvent pas être consommés au cours de cette sous-séquence et sont donc présents lors du tir de  $t''(k)$ .

Réciproquement, soit  $\sigma'$  une séquence de tir acceptante infinie de  $\mathcal{N}'$ . Nous obtenons une séquence  $\sigma$  de  $\mathcal{N}$  acceptant le même mot temporisé comme suit. Remarquons d'abord qu'à chaque fois qu'une transition  $t''(k)$  est franchie dans  $\sigma'$ , nous pouvons choisir de consommer le jeton le plus ancien présent dans  $p_2$  d'âge inférieur ou égal à  $\frac{a}{2}$  sans perdre le fait que  $\sigma$  peut être tirée (en effet les jetons de la place  $p_2$  sont toujours testés par des intervalles de la forme  $[0, x)$ ). Nous supposons donc un tel comportement.

Nous effaçons simplement les occurrences de la transition de transfert et les occurrences des transitions contrôlant le comportement cyclique (*i.e.* celles apparaissant dans la partie inférieure du réseau). Nous substituons également à  $t'(n'_1, k)$  (respectivement  $t''(k)$ ) la transition  $t'$  (respectivement  $t''$ ). Il reste à définir l'âge initial d'un jeton produit dans la place  $p$ . Pour cela, nous distinguons trois cas. Si ce jeton correspond dans  $\mathcal{N}'$  à un jeton qui n'est jamais transféré dans  $p_2$ , alors son âge initial est  $\frac{a}{2}$ . Si le jeton correspondant est transféré dans  $p_2$  lorsque son âge vaut  $\delta$ , mais n'est jamais consommé par une certaine transition  $t''(k)$ , alors dans  $\mathcal{N}$  son âge initial est défini par  $\frac{a}{2} - \delta$ . Enfin, si ce jeton est transféré dans la place  $p_2$  avec un âge  $\delta$  et est consommé par une certaine transition  $t''(k)$  lorsque son âge vaut  $\delta'$ , alors son âge initial dans  $\mathcal{N}$  est égal à  $a - \delta - \delta'$ . Remarquons que ce dernier choix implique que la transition  $t''$  est encore franchissable dans  $\mathcal{N}$ .

Nous devons alors vérifier que les définitions présentées précédemment pour l'âge initial des jetons produits sont compatibles avec le tir de la transition  $t'$ . Considérons donc une occurrence dans  $\sigma$  d'un arc de lecture dont l'étiquette vaut  $n' \cdot ]0, a[$ . Afin d'être franchissable, cet arc de lecture requiert la présence de  $n'$  jetons dans  $p$  d'âge inférieur à  $a$ . Ce test correspond dans  $\mathcal{N}'$  au tir d'une transition  $t'(n'_1, k)$  avec  $n' - n'_1 \leq k$  au cours d'une certaine séquence de tir instantanée  $\rho$ . Les  $n'_1$  jetons de  $p$  utilisés par ce tir ont, par construction, un âge inférieur à  $a$  (notons que ces jetons peuvent éventuellement être transférés dans  $p_2$  après un certain délai). Considérons à présent les  $n' - n'_1$  plus jeunes jetons dans la place  $p_2$  au début de  $\rho$ . Nous allons démontrer qu'ils ont tous un âge strictement inférieur à  $a$  dans  $\mathcal{N}$ . D'abord, remarquons qu'aucun d'entre eux ne peut être consommé par une transition  $t''$  au cours de  $\rho$  puisqu'un tir de  $t''$  requiert au moins  $k \geq n' - n'_1$  jetons en plus du jeton à consommer, et car nous avons supposé que les transitions  $t''$  consomment toujours les jetons les plus âgés. Considérons à présent un de ces jetons. Deux cas peuvent se produire. Ou bien ce jeton sera consommé plus tard (*i.e.* dans une autre séquence instantanée) par une transition  $t''(k)$ , et alors son âge est strictement inférieur à  $a$ . Ou bien ce jeton ne sera jamais consommé, et alors son âge dans  $\mathcal{N}'$  est égal à un certain  $\delta' < \frac{a}{2}$ . D'après les définitions précédentes de l'âge initial d'un jeton produit

dans  $p$  dans  $\mathcal{N}$ , l'âge affecté à ce jeton est  $\frac{a}{2} + \delta'$ , qui vérifie donc bien  $\frac{a}{2} + \delta' < a$ .

Ceci conclut la preuve pour ce second motif  $\mathcal{P}_5$ .  $\square$

## 5.7 Synthèse et applications

Dans cette section, nous dressons d'abord un bilan des résultats obtenus dans les précédentes sections. Ensuite, nous rapprochons ces résultats de l'équivalence avec les automates temporisés établie dans la sous-section 5.2.2. Ceci nous permet d'obtenir des résultats sur la comparaison des deux modèles, ainsi que de nouveaux résultats concernant le modèle des automates temporisés.

### 5.7.1 Synthèse des résultats d'expressivité

Nous présentons ici une synthèse des différents résultats obtenus au cours des études précédentes. De plus, pour améliorer la lisibilité des résultats, nous utilisons les abréviations suivantes pour les différentes classes étudiées :

- RDPT pour la classe des réseaux de Petri temporisés (sans arcs de lecture),
- RDPTLEC pour la classe des réseaux de Petri temporisés avec arcs de lecture,
- le préfixe 0- pour représenter la restriction de la classe aux réseaux n'utilisant que des remises à zéro,
- le préfixe « intégral- » pour représenter la restriction de la classe aux réseaux intégraux.

**Cas des mots finis et infinis non Zeno.** Appliquant les résultats démontrés dans les sous-sections précédentes, nous obtenons l'égalité de toutes les sous-classes des réseaux de Petri temporisés avec arcs de lecture mentionnées sur la figure 5.26, du point de vue de l'équivalence de langages de mots finis et infinis non Zeno. Remarquons que ces égalités sont correctes pour les classes générales, mais aussi pour leurs restrictions aux réseaux bornés et/ou à caractère intégral. D'autre part, nous avons indiqué sous les égalités les références des théorèmes permettant de les obtenir.

$$\text{RDPTLEC} \underbrace{\equiv_{*,\omega_{nz}}}_{\text{Théo. 5.21,5.22}} \text{RDPT} \underbrace{\equiv_{*,\omega_{nz}}}_{\text{Théo. 5.24}} \text{0- RDPT}$$

FIG. 5.26 – Expressivité relative des réseaux de Petri temporisés avec arcs de lecture pour les mots finis et infinis non Zeno

**Cas des mots infinis.** La situation du point de vue de l'équivalence des mots infinis est nettement différente. En effet la hiérarchie s'écrasait dans le cas précédent, tandis que dans ce cas nous obtenons le treillis représenté sur la figure 5.27. Les arcs pleins représentent des inclusions strictes, tandis que les arcs pointillés signifient que les classes sont incomparables. De plus, nous avons indiqué sur les arcs pleins l'argument permettant de démontrer l'inclusion stricte entre les deux classes (restriction à des réseaux intégraux, ou un des deux langages discriminants de la sous-section 5.4.1). Comme précédemment, nous indiquons les références des théorèmes permettant de prouver les égalités. Enfin, cette figure, valable pour les classes générales, est encore correcte pour leurs restrictions aux réseaux bornés.



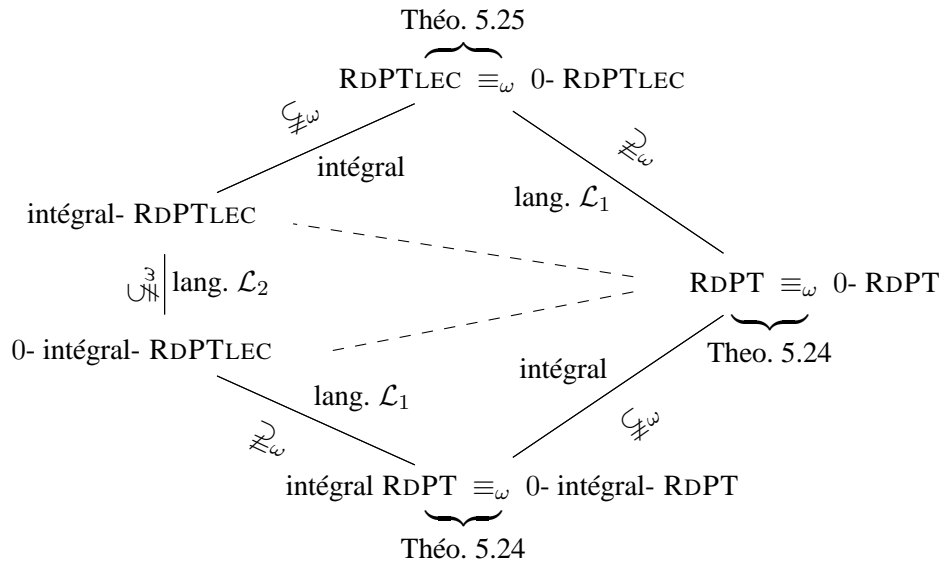


FIG. 5.27 – Expressivité relative des réseaux de Petri temporisés avec arcs de lecture pour les mots infinis

### 5.7.2 Applications aux automates temporisés

L'équivalence avec les automates temporisés démontrée dans la sous-section 5.2.2 (Théorème 5.6) permet de déduire des résultats obtenus précédemment de nouveaux résultats sur les liens entre les automates temporisés et les réseaux de Petri temporisés. Nous obtenons ainsi les résultats recherchés sur la comparaison des pouvoirs expressifs des automates temporisés et des réseaux de Petri temporisés. De plus, nous obtenons également « gratuitement » de nouveaux résultats précisant le rôle des mises à jour non déterministes dans les automates temporisés.

**Corollaire 5.26.** *Pour les équivalences de mots finis et infinis non Zeno,*

- (1) *les réseaux de Petri temporisés bornés et les automates temporisés à mises à jour non déterministes sont aussi expressifs,*
- (2) *les automates temporisés avec mises à jour non déterministes et les automates temporisés restreints aux remises à zéro sont aussi expressifs. De plus, imposer aux automates le caractère intégral préserve cette équivalence.*

**Corollaire 5.27.** *Pour l'équivalence de mots infinis,*

- (3) *les réseaux de Petri temporisés et les automates temporisés sont incomparables<sup>3</sup>,*
- (4) *les automates temporisés sont strictement plus expressifs que les réseaux de Petri temporisés bornés,*
- (5) *les automates temporisés à caractère intégral et à mises à jour non déterministes sont strictement plus expressifs que les automates temporisés à caractère integral (et à remises à zéro),*

<sup>3</sup>Rappelons que les réseaux de Petri peuvent reconnaître des langages non réguliers.

- (6) *les automates temporisés à mises à jour non déterministes et les automates temporisés (à remises à zéro) sont aussi expressifs.*

Il était communément admis que les automates temporisés et les réseaux de Petri temporisés bornés étaient aussi expressifs. Nous avons montré que c'est effectivement le cas pour les équivalences de mots finis et infinis non Zeno (point (1)), mais que cela est faux lorsque les comportements Zeno sont pris en compte (point (4)). En réalité, ce résultat est même plus fort : bien que les réseaux de Petri temporisés peuvent être vus comme des systèmes temporisés manipulant un nombre infini d'horloges, nous avons démontré que les automates temporisés et les réseaux de Petri temporisés sont en toute généralité incomparables (point (3)).

Les trois autres résultats complètent les résultats existants sur le rôle des mises à jour non déterministes dans les automates temporisés [BDFP04] rappelés dans le théorème 1.24 page 43. Le point (2) a déjà été partiellement prouvé dans [BDFP04] et nous fournissons ici une nouvelle démonstration de ce résultat. Les points (5) et (6) sont assez surprenants puisqu'ils montrent qu'il est nécessaire de raffiner la granularité des gardes afin de s'affranchir des mises à jour non déterministes dans les automates temporisés (afin de préserver les langages de mots infinis). À notre connaissance, ceci constitue un des premiers résultats de ce type dans le cadre des systèmes temporisés. Enfin, la transformation que nous avons développée pour la preuve du théorème 5.25 peut être appliquée aux automates temporisés et étend ainsi au cadre des mots infinis la construction présentée dans [BDFP04] permettant de retirer les mises à jour non déterministes (la construction de [BDFP04] est en fait correcte seulement pour les mots finis et infinis non Zeno). Nous illustrons notre transformation sur la figure 5.28 en exhibant un automate temporisé (à remises à zéro)  $\omega$ -équivalent à l'automate temporisé représenté sur la figure 5.7(b).

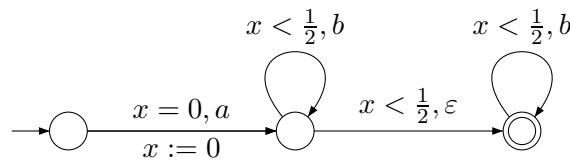


FIG. 5.28 – Un exemple de la transformation permettant de s'affranchir des mises à jour non déterministes pour les automates temporisés.

## 5.8 Conclusion

Dans ces travaux, nous avons étudié les pouvoirs expressifs relatifs des réseaux de Petri temporisés et des automates temporisés. Nous avons démontré en particulier que, contrairement à l'intuition communément admise, ces deux modèles sont en toute généralité incomparables. Ceci donne un rôle très intéressant au modèle des réseaux de Petri temporisés avec arcs de lecture que nous avons étudié ici puisqu'il fournit un cadre unifiant les deux modèles précédents. De plus, nous avons prouvé qu'il possède une propriété intéressante, la décidabilité du problème de couverture.

Nous nous sommes ensuite intéressés à des questions d'expressivité. Plus précisément, nous avons d'abord cherché à répondre aux questions suivantes, posées dans l'introduction :

- est-il nécessaire d'étendre le modèle des réseaux de Petri temporisés afin de pouvoir exprimer les automates temporisés ?

- les arcs de lecture augmentent-ils le pouvoir d'expression des réseaux de Petri temporisés ?
- que deviennent ces questions sous les contraintes de réseaux bornés, vis-à-vis des différentes équivalences de langages (mots finis, infinis, infinis non Zeno) ?

Les réponses à ces questions s'avèrent être relativement complexes car elles dépendent de l'équivalence prise en compte pour comparer les modèles. Ainsi, en toute généralité, les réponses aux deux premières questions sont positives. Cependant, vis-à-vis des équivalences de mots temporisés finis et de mots temporisés infinis non Zeno, les deux modèles des automates temporisés bornés et des réseaux de Petri temporisés bornés (et même saufs) sont aussi expressifs et les réponses deviennent donc négatives. Nuancions tout de même ce résultat car il repose sur des transformations complexes tandis que le modèle des réseaux de Petri temporisés avec arcs de lecture permet lui d'exprimer très simplement les automates temporisés. La transformation préserve en effet la taille du modèle. Finalement, soulignons que ce sont les comportements Zeno, souvent écartés lors de considérations de modélisation car ils ne peuvent pas correspondre à des comportements réels d'un système, qui permettent de distinguer les deux modèles.

Dans une seconde étape, nous nous sommes intéressés au rôle des mises à jour non déterministes. En effet, alors que dans le modèle standard des automates temporisés, seules des mises à zéro sont considérées, les réseaux de Petri temporisés font en général intervenir des mises à jour non déterministes nettement plus générales. Nous avons démontré que ces deux classes ont le même pouvoir d'expression, mais que la granularité du modèle doit être raffinée afin de pouvoir s'affranchir des mises à jour non déterministes dans les réseaux de Petri temporisés avec arcs de lecture et en présence de comportements Zeno. À nouveau, ce sont donc ces mêmes comportements qui discriminent les deux familles de modèles. À notre connaissance, ceci constitue un des premiers résultats dans le domaine des systèmes temporisés qui exprime la nécessité d'un raffinement de la granularité du modèle. De plus, ces résultats permettent de compléter (et même corriger) le travail de [BDFP04] sur le rôle des mises à jour non déterministes dans les automates temporisés.

Les prolongements de ces travaux sont divers. En termes d'expressivité d'abord, il est naturel de s'intéresser à l'ajout d'urgence au modèle des réseaux de Petri temporisés, qui en est démuné dans ces travaux. Il est déjà connu que le modèle (non borné) devient indécidable avec une sémantique d'urgence forte, mais l'introduction d'une urgence plus faible au travers d'invariants sur les places pourrait conserver les propriétés de décidabilité. Ceci permettrait également d'étendre notre comparaison au cadre des automates temporisés avec invariants. En termes de décidabilité, il serait intéressant de chercher à étendre les résultats de décidabilité existant dans le cadre des réseaux de Petri temporisés [AMM07] au cadre du modèle avec arcs de lecture. Enfin, une application fondamentale de ces travaux au cadre de la vérification est l'utilisation des idées mises en œuvre dans les transformations proposées pour le développement de techniques de dépliages pour les réseaux d'automates temporisés. Ceci fait l'objet du chapitre 7.



**Troisième partie**

**Algorithmes pour la Vérification**



## Chapitre 6

# Analyse en avant des automates temporisés avec contraintes diagonales

Dans ce chapitre, nous étudions l’extension de l’algorithme d’analyse en avant à la volée au modèle des automates temporisés avec contraintes diagonales. Il a été démontré dans [Bou04] que l’algorithme classique n’est pas correct pour cette classe. Nous détaillons l’origine des problèmes, proposons une solution utilisant le paradigme du raffinement basé sur l’analyse de contre-exemples et évaluons cette solution à l’aide d’une implémentation dans l’outil UPPAAL. Les résultats présentés ici ont été publiés dans [BLR05] et l’implémentation est décrite dans [Rey07].

### 6.1 Introduction

Nous avons déjà présenté dans le chapitre 1 le modèle des automates temporisés ainsi que son extension avec des contraintes diagonales dans la sous-section 1.5.3. Nous avons rappelé que l’expressivité de ces deux modèles est la même, mais que le second est exponentiellement plus concis que le premier. De plus, les contraintes diagonales facilitent la modélisation de systèmes temporisés.

La décidabilité de l’accessibilité dans les automates temporisés a été obtenue via la construction de l’automate des régions (section 1.4.3). Nous avons montré que la taille de cet automate est exponentielle en la taille de l’automate temporisé et sa construction explicite n’est donc pas efficace en pratique. Des algorithmes à la volée ont été proposés afin de pallier cette difficulté et leur efficacité a été largement démontrée puisqu’ils ont permis d’analyser des systèmes réels de taille importante [HSL97, TY98].

Il a récemment été démontré dans [Bou04] que l’algorithme d’analyse à la volée en avant, pourtant implémenté depuis plusieurs années et utilisé en pratique, est incorrect pour le cadre des automates temporisés avec gardes diagonales dans [Bou04]. Ceci a constitué une surprise importante dans la communauté scientifique puisque la correction de cet algorithme était communément admise. Cependant, ces travaux démontrent également que pour la classe des automates temporisés sans gardes diagonales, l’algorithme implémenté dans les outils UPPAAL et KRONOS est correct. Ceci explique en partie pourquoi ce « bug » dans un algorithme aussi classique et répandu n’a pas été découvert plus tôt : peu de modèles parmi ceux étudiés jusqu’alors utilisent des gardes diagonales et de plus les modèles provoquant une erreur de l’algorithme sont encore plus rares.

Comme nous l’avons rappelé précédemment, les gardes diagonales apportent de la concision au modèle et simplifient la modélisation. Elles ont par exemple été utilisées dans le cadre de la modélisation de problèmes d’ordonnancement de tâches [FPY02]. Il existe également une version du protocole

de Fischer temporisé, introduite dans [Zbr05], qui utilise des gardes diagonales. Il est donc intéressant de proposer des solutions permettant de corriger ce « bug » sans augmenter sensiblement la complexité de l'algorithme. De plus l'algorithme classique, grâce à son approche « en avant » peut traiter des systèmes contenant des variables entières, comme cela est fait dans l'outil UPPAAL. Un calcul en arrière ne permet pas de prendre en compte ce type de variables supplémentaires. En effet, les prédécesseurs d'une affectation de la forme  $i := j.k + l.m$  où les variables  $i, j, k, l$  et  $m$  représentent des variables entières ne peuvent pas être calculés de façon symbolique. Il est donc nécessaire d'énumérer tous les uplets de valeurs ce qui n'est pas souhaitable en pratique. C'est pourquoi nous cherchons à développer un algorithme préservant cet aspect en avant.

Dans un premier temps, nous présentons l'algorithme classique d'analyse en avant pour le modèle des automates temporisés d'Alur et Dill (section 6.2). Nous étudions ensuite l'impact sur cet algorithme de l'introduction des contraintes diagonales (section 6.3). Nous analysons dans la section 6.4 des solutions possibles afin de « corriger » l'algorithme classique motivant ainsi le choix d'une méthode fondée sur le raffinement successif du modèle guidé par l'analyse des contre-exemples. Ceci nous conduit à développer dans la section 6.5 le cœur de notre solution, l'analyse des contre-exemples afin de détecter les gardes diagonales « responsables » des erreurs commises par l'algorithme. Nous présentons enfin l'implémentation de cette méthode dans l'outil UPPAAL ainsi qu'une série de tests afin de démontrer son efficacité (section 6.6) et donnons quelques conclusions et perspectives dans la section 6.7.

Notons enfin que dans la suite de ce chapitre, nous nous plaçons dans la classe des automates temporisés à constantes entières et sans invariants. Nous avons déjà discuté cette restriction dans la sous-section 1.4.2 et montré qu'elle n'est pas restrictive du point de vue du problème de l'accessibilité d'un état de contrôle. De plus, les algorithmes que nous présentons dans ce chapitre s'adaptent au cadre général des automates temporisés et la restriction précédente permet seulement de simplifier largement cette présentation.

## 6.2 Analyse en avant des automates temporisés

Nous présentons dans cette section l'algorithme standard d'analyse en avant à la volée des automates temporisés. L'analyse en avant est une technique classique pour décider de l'accessibilité d'un ensemble d'états cibles. Elle consiste à calculer itérativement les successeurs en un pas des configurations initiales jusqu'à, ou bien atteindre un état cible, ou bien converger. L'algorithme 1 présente cette méthode pour le cadre d'un système de transitions. En toute généralité, cet algorithme peut ne pas terminer si le système est infini et c'est en particulier le cas pour les automates temporisés. Nous utilisons donc une abstraction finie du système fondée sur une représentation symbolique des ensembles de valuations d'horloges, que nous décrivons dans la sous-section 6.2.1. Nous présentons ensuite la structure de données permettant d'implémenter cette représentation symbolique dans la sous-section 6.2.2. Enfin, nous donnons les éléments principaux de la preuve de correction de cet algorithme dans la sous-section 6.2.3.

### 6.2.1 Les zones, une représentation symbolique de valuations

Afin de manipuler plus efficacement les ensembles de valuations d'horloges calculés lors de l'analyse de l'automate temporisé, il est nécessaire de disposer d'une représentation symbolique de ces ensembles. Cette représentation doit être compatible avec le calcul des successeurs. Elle doit également être aisément implémentable et être manipulable de façon efficace, *i.e.* que les opérations précédentes



---

**Algorithme 1** Analyse en avant à la volée pour un système de transitions.
 

---

**Entrée :**  $\mathcal{S} = (Q, q_0, \rightarrow)$  un système de transitions,  $q_f \in Q$  un état cible

**Sortie :** Réponse au problème de l'accessibilité de  $q_f$  dans  $\mathcal{S}$ 

```

1: Visités :=  $\emptyset$ ; /* Initialisation */
2: Attente :=  $\{q_0\}$ ;
3: Répéter
4:   Piocher  $q$  dans Attente ;
5:   Si  $q \notin$  Visités Alors
6:     Calculer Successeurs =  $\{q' \in Q \mid q \rightarrow q'\}$ ;
7:     Si  $q_f \in$  Successeurs Alors
8:       Retourner « OUI » ;
9:     Sinon /* Successeurs ne contient pas  $q_f$  */
10:      Visités := Visités  $\cup \{q\}$ ;
11:      Attente := Attente  $\cup$  Successeurs ;
12:    Fin Si
13:  Fin Si
14: Jusqu'à Attente =  $\emptyset$ ;
15: Retourner « NON » ;

```

---

doivent être de faible complexité. Enfin, cette représentation symbolique doit avoir un ensemble fini de représentants afin de garantir la terminaison de l'algorithme.

Les régions, introduites dans la sous-section 1.4.3, constituent une représentation symbolique satisfaisant ces différents critères. Cependant, le nombre de régions est exponentiel dans la taille de l'automate et il est possible de manipuler des ensembles plus grands de valuations d'horloges afin de réduire le nombre d'objets construits par l'algorithme. Ceci est obtenu à l'aide de la représentation symbolique des *zones*, qui est utilisée dans la majorité des algorithmes développés pour les systèmes temporisés.

**Définition 6.1 (Zone).** *Étant donné un ensemble d'horloges  $X$ , une zone  $Z$  sur cet ensemble  $X$  est l'ensemble des valuations sur  $X$  vérifiant une contrainte d'horloges  $\varphi \in \mathcal{C}(X)$  définie uniquement par des conjonctions, i.e.  $Z = \llbracket \varphi \rrbracket$ . L'ensemble des contraintes d'horloges admissibles pour la définition d'une zone, appelées contraintes de zone, est décrit par la grammaire suivante :*

$$\varphi ::= x \sim c \mid x - y \sim c \mid \varphi \wedge \varphi \mid \mathbf{tt}$$

où  $x, y \in X, c \in \mathbb{N}, \sim \in \{<, \leq, =, \geq, >\}$ .

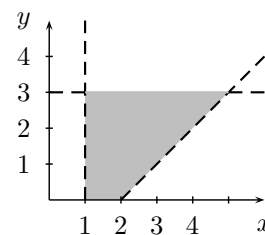
Enfin, étant donné un entier naturel  $k \in \mathbb{N}_{>0}$ , une zone est dite *k-bornée* si et seulement si elle peut être obtenue par une contrainte d'horloge *k-bornée*.

Notons qu'une zone constitue toujours un ensemble de valuations convexe.

**Exemple 6.2.** La zone illustrée sur la figure ci-contre est associée à la contrainte suivante :

$$\varphi = x > 1 \wedge y \leq 3 \wedge x - y < 2$$

L'horloge  $x$  est placée en abscisse et l'horloge  $y$  en ordonnée.



┘

Considérons à présent un automate temporisé  $\mathcal{A}$ , d'espace d'états de contrôle  $L$  et d'horloges  $X$ . Nous définissons un *état symbolique* de  $\mathcal{A}$  comme une paire  $(\ell, Z)$  où  $\ell \in L$  et  $Z$  est une zone sur  $X$ .

Nous définissons l'opérateur **Post** sur les zones de la façon suivante. Il combine à la fois le calcul des successeurs discrets par une transition donnée et le calcul des successeurs temporels. Rappelons que nous avons supposé que  $\mathcal{A}$  ne possède pas d'invariants. Soit  $(\ell, Z)$  un état symbolique et  $t = \ell \xrightarrow{g, a, R} \ell'$  une transition de l'automate temporisé. Alors la zone<sup>1</sup>  $\text{Post}(Z, t)$  est définie par :

$$\text{Post}(Z, t) = \{v[R \leftarrow 0] + d \mid d \in \mathbb{T}, v \in Z, v \models g\}$$

$(\ell', \text{Post}(Z, t))$  est l'état symbolique atteint depuis l'état symbolique  $(\ell, Z)$  à l'issue du franchissement de la transition discrète  $t$ , et après écoulement du temps.

Nous présentons maintenant les opérations de base réalisables sur les zones. Afin de calculer les successeurs possibles d'une configuration d'un automate temporisé, nous devons considérer les successeurs discrets, obtenus par intersection avec la contrainte étiquetant la transition correspondante puis par application de la remise à zéro et enfin calculer les successeurs temporels de la nouvelle configuration. Les zones constituent une structure stable par chacune de ces trois opérations (intersection, remise à zéro et successeurs temporels) comme cela est énoncé dans la proposition suivante :

**Proposition 6.3** (Opérations sur les zones). *Considérons deux zones  $Z_1$  et  $Z_2$  sur un ensemble d'horloges  $X$  et un sous-ensemble d'horloges  $R \subseteq X$ . Alors les trois ensembles suivants sont encore des zones sur  $X$  :*

**Intersection :** l'ensemble  $Z_1 \cap Z_2$ ,

**Remise à zéro :** l'ensemble  $\{v[R \leftarrow 0] \mid v \in Z_1\}$ , noté  $Z_1[R \leftarrow 0]$ ,

**Futur :** l'ensemble  $\{v + d \mid v \in Z_1, d \in \mathbb{T}\}$ , noté  $\overrightarrow{Z_1}$ .

Notons que l'ensemble  $\text{Post}(Z, t)$  peut être défini par l'égalité suivante :

$$\text{Post}(Z, t) = \overrightarrow{(Z \cap \llbracket g \rrbracket)[R \leftarrow 0]}$$

D'après la proposition précédente,  $\text{Post}(Z, t)$  est donc bien une zone dès lors que  $g$  est une zone.

De plus, l'ensemble constitué de la valuation initiale  $\mathbf{0}$  est également une zone ( $\bigwedge_{x \in X} x = 0$ ), notée par la suite  $Z_0$ .

Il est donc possible de résoudre dans ce formalisme le problème de l'accessibilité d'un état de contrôle défini dans le chapitre 1 en adaptant l'algorithme 1. Cependant, le nombre de zones n'est *a priori* pas borné et l'algorithme peut donc (encore) ne pas terminer. Afin de pallier cette difficulté, nous nous restreignons à un sous-ensemble fini de l'ensemble des zones sur  $X$ , l'ensemble des zones  $K$ -bornées, où  $K$  dénote la constante maximale apparaissant dans l'automate étudié  $\mathcal{A}$ . Afin de garantir que les zones manipulées sont toujours  $K$ -bornées, nous utilisons un opérateur d'extrapolation  $\text{Approx}_K$  défini ainsi :

$$\text{Approx}_K(Z) = \bigcap_{\substack{\text{zones } K\text{-bornées } Z' \\ \text{telles que } Z \subseteq Z'}} Z'$$

Il est facile de vérifier que cette définition est correcte et que la zone  $\text{Approx}_K(Z)$  ainsi définie est  $K$ -bornée, contient la zone  $Z$  et est la plus petite zone, au sens de l'inclusion, vérifiant ces deux

<sup>1</sup>Afin de garantir que cet objet est une zone, il est nécessaire de supposer que  $g$  en est une, *i.e.* qu'il faut avoir exclu les disjonctions de la définition des gardes des transitions.

---

**Algorithme 2** Analyse en avant à la volée pour les automates temporisés.
 

---

**Entrée :**  $\mathcal{A}$  un automate temporisé,  $\ell_f$  un état de contrôle cible

**Sortie :** Réponse au problème de l'accessibilité de  $\ell_f$  dans  $\mathcal{A}$ 

```

1: Calculer la constante maximale  $K$  apparaissant dans  $\mathcal{A}$ ;
2: Visités :=  $\emptyset$ ; /* Initialisation */
3: Attente :=  $\{(\ell_0, \vec{Z}_0)\}$ ;
4: Répéter
5:   Piocher  $(\ell, Z)$  dans Attente;
6:   Si il n'existe pas  $(\ell, Z') \in$  Visités tel que  $Z \subseteq Z'$  Alors /*  $(\ell, Z)$  n'a pas encore été visité */
7:     Successeurs :=  $\{(\ell', \text{Approx}_K(\text{Post}(Z, t))) \mid t = \ell \xrightarrow{g, a, R} \ell' \text{ transition de } \mathcal{A}\}$ ;
8:     Si il existe  $(\ell_f, Z_f) \in$  Successeurs tel que  $\llbracket Z_f \rrbracket \neq \emptyset$  Alors
9:       Retourner « OUI »;
10:    Sinon /* Successeurs ne contient pas d'état final */
11:      Visités := Visités  $\cup \{(\ell, Z)\}$ ;
12:      Attente := Attente  $\cup$  Successeurs;
13:    Fin Si
14:  Fin Si
15: Jusqu'à Attente =  $\emptyset$ ;
16: Retourner « NON »;

```

---

propriétés. L'opérateur  $\text{Approx}_K$  calcule donc une surapproximation de la zone  $Z$ , *i.e.* pour toute zone  $Z$ , nous avons l'inclusion suivante :

$$Z \subseteq \text{Approx}_K(Z) \quad (6.1)$$

Les éléments précédents sur les zones nous permettent de présenter maintenant l'algorithme 2 qui constitue l'algorithme standard d'accessibilité en avant et à la volée pour les automates temporisés (sans gardes diagonales). Cet algorithme est une simple adaptation de l'algorithme 1 présenté plus haut.

L'inclusion (6.1) implique que l'algorithme 2 calcule une surapproximation de l'ensemble des états accessibles. Plus précisément, rappelons que nous notons  $\text{D-Acc}(\mathcal{A})$  l'ensemble des états de contrôle de  $\mathcal{A}$  qui sont accessibles dans  $\mathcal{A}$ . De façon similaire, définissons  $\text{D-Acc}(\mathcal{A}, \text{Approx}_K)$  l'ensemble des états de contrôle calculés comme accessibles par l'algorithme 2 dépendant de l'opérateur d'extrapolation  $\text{Approx}_K$ . D'après la propriété 6.1, nous avons :

$$\text{D-Acc}(\mathcal{A}) \subseteq \text{D-Acc}(\mathcal{A}, \text{Approx}_K) \quad (6.2)$$

Nous démontrerons dans la sous-section 6.2.3 la correction de l'algorithme 2 pour la classe des automates temporisés sans gardes diagonales. Celle-ci s'exprime ainsi :

$$\forall \ell \in L, \ell \in \text{D-Acc}(\mathcal{A}) \iff \ell \in \text{D-Acc}(\mathcal{A}, \text{Approx}_K) \quad (6.3)$$

Avant cela, nous présentons la structure de données utilisée pour implémenter les zones.

**Remarque 6.4.** Notons que le nombre de zones  $K$ -bornées est, comme les régions, exponentiel dans la taille de l'automate  $\mathcal{A}$ . Plus précisément, il est même plus grand que le nombre de régions. Ainsi, le nombre d'éléments de la représentation symbolique n'a pas été réduit par rapport à la représentation des régions. Cependant, les ensembles peuvent être plus grands, du point de vue de l'inclusion, et l'algorithme résultant est beaucoup plus efficace en pratique. ┘

## 6.2.2 Matrices à différences bornées

Nous présentons dans cette sous-section la structure de données habituellement utilisée afin de représenter et manipuler les zones lors de l'implémentation de cet algorithme. Les opérations nécessaires pour l'implémentation de l'algorithme 2 sont les suivantes : test d'inclusion, test du vide, calcul de l'image par Post et par  $\text{Approx}_k$ . Cette structure de données, intitulée matrices à différences bornées, a été introduite et étudiée dans [BM83, Dil90].

**Définition 6.5** (Matrice à différences bornées [BM83, Dil90]). *Une matrice à différences bornées pour un ensemble  $X$  de  $n$  horloges est une matrice carrée de dimension  $(n+1) \times (n+1)$  contenant des paires  $(\prec, m)$  éléments de l'ensemble  $\mathbb{V} = \{<, \leq\} \times \mathbb{Z} \cup \{(<, \infty)\}$ . Nous notons les  $n$  horloges considérées  $x_1, \dots, x_n$  et considérons une horloge supplémentaire  $x_0$ . L'horloge  $x_0$  sert seulement de référence, et sa valeur dans la sémantique sera toujours 0.*

À une matrice à différences bornées  $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$  est associé le sous-ensemble  $\llbracket M \rrbracket$  de  $\mathbb{T}^X$  défini par  $\llbracket M \rrbracket = \llbracket \varphi \rrbracket$  où  $\varphi$  est une contrainte de zone sur  $X$  définie par :

$$\varphi = \bigwedge_{0 \leq i,j \leq n} x_i - x_j \prec_{i,j} m_{i,j}$$

où le terme  $x_0$  est remplacé par 0. Ceci implique donc que  $\llbracket M \rrbracket$  est une zone sur  $X$ .

En particulier, il est facile de vérifier qu'étant donnée une zone  $Z = \llbracket \varphi \rrbracket$  sur  $X$ ,  $\varphi$  étant une contrainte de zone sur  $X$ , nous pouvons définir une matrice à différences bornées  $M$  vérifiant  $\llbracket M \rrbracket = Z$ . Cependant, cette matrice n'est pas unique, comme cela est illustré sur l'exemple 6.6.

**Exemple 6.6** (Matrices à différences bornées et zones.). La zone définie par les équations  $x_1 > 3 \wedge x_2 \leq 5 \wedge x_1 - x_2 < 4$  peut être représentée par les deux matrices à différences bornées suivantes :

$$\left( \begin{array}{ccc} (\leq, 0) & (<, -3) & (<, \infty) \\ (<, \infty) & (\leq, 0) & (<, 4) \\ (\leq, 5) & (<, \infty) & (\leq, 0) \end{array} \right) \text{ et } \left( \begin{array}{ccc} (<, \infty) & (<, -3) & (<, \infty) \\ (\leq, \infty) & (<, \infty) & (<, 4) \\ (\leq, 5) & (<, \infty) & (\leq, 0) \end{array} \right)$$

┘

**Forme normale.** Avec cette définition, plusieurs matrices à différences bornées peuvent définir la même zone. Ceci pose problème car il n'est alors pas possible, étant données deux matrices à différences bornées  $M_1$  et  $M_2$ , de tester de manière syntaxique si elles représentent la même zone, *i.e.* si  $\llbracket M_1 \rrbracket = \llbracket M_2 \rrbracket$ . Une *forme normale* a donc été introduite afin de représenter sans ambiguïté les zones par des matrices à différences bornées. La forme normale d'une matrice à différences bornées donnée est simplement l'unique matrice à différences bornées qui, parmi celles qui représentent la même zone, a les coefficients les plus petits pour l'ordre défini ci-dessous. Cette matrice à différences bornées « minimale » représente les contraintes les plus fortes sur les horloges  $x_i$ . Une fois définis un ordre et une addition sur les éléments de  $\mathbb{V}$ , nous pouvons utiliser un algorithme classique de plus court chemin, par exemple l'algorithme de Floyd présenté au travers de l'algorithme 3, pour calculer la forme normale. Nous noterons dans la suite  $\phi(M)$  la matrice à différences bornées sous forme normale associée à la matrice à différences bornées  $M$ .

**Ordre sur  $\mathbb{V}$  :** Si  $(\prec, m), (\prec', m') \in \mathbb{V}$ , alors nous avons  $(\prec, m) \leq (\prec', m')$  si et seulement si :

$$\left\{ \begin{array}{l} m < m' \\ \text{ou} \\ m = m' \text{ et } (\prec = \prec' \text{ ou } \prec' = \leq). \end{array} \right.$$

Addition sur  $\mathbb{V}$  : Si  $(\prec, m), (\prec', m') \in \mathbb{V}$ , alors nous définissons  $(\prec'', m'') = (\prec, m) + (\prec', m')$  par :

$$\begin{cases} m'' = m + m' \\ \text{et} \\ \prec'' = \leq \text{ si } \prec = \prec' = \leq \text{ et } < \text{ sinon.} \end{cases}$$

---

**Algorithme 3** Algorithme de Floyd permettant de calculer la forme normale

---

**Entrée :**  $M = (M_{i,j})_{0 \leq i,j \leq n}$  une matrices à différences bornées.

**Sortie :**  $\phi(M)$  la forme normale de  $M$

- 1: **Pour**  $k = 0$  à  $n$  **Faire**
  - 2:     **Pour**  $i = 0$  à  $n$  **Faire**
  - 3:         **Pour**  $j = 0$  à  $n$  **Faire**
  - 4:              $M_{i,j} := \min(M_{i,j}, M_{i,k} + M_{k,j})$ ;
  - 5:         **Fin Pour**
  - 6:     **Fin Pour**
  - 7: **Fin Pour**
  - 8: **Retourner**  $M$ ;
- 

**Propriétés de la forme normale.** La forme normale  $\phi(M)$  d'une matrice à différences bornées  $M$  satisfait les propriétés suivantes :

*Correction :*  $\llbracket \phi(M) \rrbracket = \llbracket M \rrbracket$

*Unicité :*  $\llbracket M \rrbracket = \llbracket M' \rrbracket \Rightarrow \phi(M) = \phi(M')$

*La forme normale représente les contraintes les plus fortes :*

$$\phi(M) \leq M, \text{ i.e. } \forall 0 \leq i, j \leq n, \phi(M)_{i,j} \leq M_{i,j}$$

*Inclusion :*  $\llbracket M \rrbracket \subseteq \llbracket M' \rrbracket \Leftrightarrow \phi(M) \leq M' \Leftrightarrow \phi(M) \leq \phi(M')$

En particulier, remarquons que le test d'inclusion entre deux matrices à différences à bornées se fait donc en temps quadratique (comparer les coefficients un à un) dès lors que les deux matrices sont en forme normale.

**Test du vide.** La forme normale permet de plus de tester du vide. En effet, soit  $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$  une matrice à différences bornées (pas nécessairement sous forme normale), alors les propriétés suivantes sont équivalentes :

(i)  $\llbracket M \rrbracket = \emptyset$

(ii) Il existe un cycle strictement négatif dans  $M$ , i.e. une séquence d'indices distincts  $(i_1, \dots, i_{l-1})$  telle que (en posant  $i_l = i_1$ )

$$(\prec_{i_1, i_2}, m_{i_1, i_2}) + \dots + (\prec_{i_{l-1}, i_l}, m_{i_{l-1}, i_l}) < (\leq, 0)$$

(iii)  $\phi(M)$  contient un terme strictement inférieur au terme  $(\leq, 0)$  sur la diagonale<sup>2</sup>

Ainsi le test du vide peut être réalisé très facilement (temps linéaire) sur une matrice à différences bornées sous forme normale puisqu'il suffit de tester les coefficients de la diagonale.

**Autres opérations utilisées par l'algorithme 2.** Il nous reste enfin à vérifier que les autres opérations nécessaires pour l'algorithme sur les zones sont aisément implémentables grâce aux matrices à différences bornées [CGP99]. Commençons par le calcul de l'image par l'opérateur **Post** et considérons ensuite la  $k$ -approximation.

---

<sup>2</sup>Si  $\llbracket M \rrbracket = \emptyset$ ,  $\phi(M)$  ne désigne plus la forme normale de  $M$ , mais le résultat de l'algorithme 3.

**Futur.** Soit  $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$  une matrice à différences bornées sous forme normale. Définissons la matrice à différences bornées  $\vec{M} = (\prec'_{i,j}, m'_{i,j})_{0 \leq i,j \leq n}$  par :

$$(\prec'_{i,j}, m'_{i,j}) = \begin{cases} (\prec_{i,j}, m_{i,j}) & \text{si } j \neq 0 \\ (<, \infty) & \text{sinon.} \end{cases}$$

Alors nous avons  $\llbracket \vec{M} \rrbracket = \llbracket M \rrbracket$  et de plus la matrice à différences bornées  $\vec{M}$  est sous forme normale.

**Intersection.** Soient  $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$  et  $M' = (\prec'_{i,j}, m'_{i,j})_{0 \leq i,j \leq n}$  deux matrices à différences bornées (pas nécessairement sous forme normale). Définissons  $M'' = (\prec''_{i,j}, m''_{i,j})_{0 \leq i,j \leq n}$  par :

$$\forall 0 \leq i, j \leq n, (\prec''_{i,j}, m''_{i,j}) = \min((\prec_{i,j}, m_{i,j}), (\prec'_{i,j}, m'_{i,j})).$$

Alors nous avons  $\llbracket M'' \rrbracket = \llbracket M \rrbracket \cap \llbracket M' \rrbracket$ . Remarquons que nous ne pouvons pas assurer que  $M''$  soit sous forme normale, même si  $M$  et  $M'$  le sont.

**Remise à zéro.** Soit  $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$  une matrice à différences bornées sous forme normale. Définissons la matrice à différences bornées  $M_{x_k:=0} = (\prec'_{i,j}, m'_{i,j})_{0 \leq i,j \leq n}$  par :

$$\begin{aligned} (\prec'_{i,j}, m'_{i,j}) &= (\prec_{i,j}, m_{i,j}) && \text{si } i, j \neq k \\ (\prec'_{k,k}, m'_{k,k}) &= (\prec'_{k,0}, m'_{k,0}) = (\prec'_{0,k}, m'_{0,k}) = (<, 0) \\ (\prec'_{i,k}, m'_{i,k}) &= (\prec_{i,0}, m_{i,0}) && \text{si } i \neq k \\ (\prec'_{k,i}, m'_{k,i}) &= (\prec_{0,i}, m_{0,i}) && \text{si } i \neq k \end{aligned}$$

Alors nous avons  $\llbracket M_{x_k:=0} \rrbracket = \llbracket M \rrbracket[x_k \leftarrow 0]$  et de plus la matrice à différences bornées  $M_{x_k:=0}$  est sous forme normale.

**k-approximation.** Soit  $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$  une matrice à différences bornées sous forme normale. Définissons la matrice à différences bornées  $\vec{M}^k = (\prec'_{i,j}, m'_{i,j})_{0 \leq i,j \leq n}$  par :

$$(\prec'_{i,j}, m'_{i,j}) = \begin{cases} (\prec_{i,j}, m_{i,j}) & \text{si } |m_{i,j}| \leq k \\ (<, \infty) & \text{si } m_{i,j} > k \\ (<, -k) & \text{si } m_{i,j} < -k \end{cases}$$

Alors nous avons  $\llbracket \vec{M}^k \rrbracket = \text{Approx}_k(\llbracket M \rrbracket)$  mais la matrice à différences bornées  $\vec{M}^k$  n'est pas nécessairement sous forme normale.

Finalement, il est donc possible d'implémenter l'algorithme 2 à l'aide de la structure de données des matrices à différences bornées.

### 6.2.3 Fondamentaux de la correction de l'algorithme

Plusieurs tentatives de preuves de la correction de cet algorithme peuvent être trouvées dans la littérature [Tri98, WT94], mais il est démontré dans [Bou04] que ces tentatives sont erronées. Nous présentons dans cette sous-section les éléments principaux de la preuve de correction de cet algorithme pour les automates temporisés sans gardes diagonales, telle qu'elle a été réalisée dans [Bou04]. Nous expliquerons par la suite pourquoi cette preuve n'est pas correcte en présence de gardes diagonales. Celle-ci fonctionne en deux étapes.

Dans un premier temps, la correction d'un algorithme semblable à l'algorithme 2 est prouvée. La seule différence réside dans l'opérateur d'extrapolation utilisé, fondé sur les régions. Cet opérateur,

noté  $\text{Cl\^o}t\text{ure}_{\mathcal{R}}$ , dépend d'un paramètre  $\mathcal{R}$  qui désigne un ensemble de régions, au sens défini dans la sous-section 1.4.3 sur l'automate des régions. Pour l'instant,  $\mathcal{R}$  dénote l'ensemble de régions introduit pour la construction de l'automate des régions d'un automate temporisé sans gardes diagonales. Nous définissons, pour toute zone  $Z$  sur  $X$ , l'opérateur  $\text{Cl\^o}t\text{ure}_{\mathcal{R}}$  par :

$$\text{Cl\^o}t\text{ure}_{\mathcal{R}}(X) = \bigcup \{R \in \mathcal{R} \mid Z \cap R \neq \emptyset\}$$

Notons  $\text{D-Acc}(\mathcal{A}, \text{Cl\^o}t\text{ure}_{\mathcal{R}})$  l'ensemble des états de contrôle calculés comme étant accessibles par l'algorithme 2 dans lequel l'opérateur  $\text{Cl\^o}t\text{ure}_{\mathcal{R}}$  est substitué à  $\text{Approx}_K$ . Il est possible de démontrer la correction de cet algorithme, en s'appuyant sur la correction de la construction de l'automate des régions [Bou04, Proposition 1].

$$\forall \ell \in L, \ell \in \text{D-Acc}(\mathcal{A}) \iff \ell \in \text{D-Acc}(\mathcal{A}, \text{Cl\^o}t\text{ure}_{\mathcal{R}}) \quad (6.4)$$

Il est important de remarquer que l'ensemble  $\text{Cl\^o}t\text{ure}_{\mathcal{R}}(Z)$  n'est pas nécessairement une zone, même si  $Z$  est une zone. En effet, cet ensemble peut ne pas être convexe. L'exemple 6.7 présente ce cas de figure. Ainsi, les objets manipulés par l'algorithme ne sont pas nécessairement des zones mais parfois des unions de zones et il s'agit alors de séparer ces unions afin de les placer dans l'ensemble Attente. En particulier, ceci implique que l'algorithme correspondant est plus coûteux, ce qui justifie qu'il n'est pas utilisé en pratique.

Dans un second temps, nous démontrons une propriété liant les opérateurs  $\text{Cl\^o}t\text{ure}_{\mathcal{R}}$  et  $\text{Approx}_K$ . Considérons un automate temporisé sans gardes diagonales  $\mathcal{A}$ . Notons  $\mathcal{R}$  l'ensemble de régions introduit dans la sous-section 1.4.3 correspondant à  $\mathcal{A}$  et  $K$  la constante maximale apparaissant dans  $\mathcal{A}$ . Nous avons alors l'inclusion suivante, pour toute zone  $Z$  sur  $X$  (ensemble des horloges de  $\mathcal{A}$ ) [Bou04, Proposition 2]

$$Z \subseteq \text{Approx}_K(Z) \subseteq \text{Cl\^o}t\text{ure}_{\mathcal{R}}(Z) \quad (6.5)$$

Nous obtenons donc les inclusions suivantes :




$$\text{D-Acc}(\mathcal{A}) \subseteq \text{D-Acc}(\mathcal{A}, \text{Approx}_K) \subseteq \text{D-Acc}(\mathcal{A}, \text{Cl\^o}t\text{ure}_{\mathcal{R}})$$

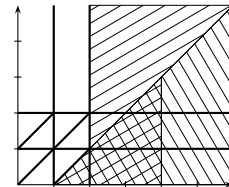
La propriété (6.4) permet d'obtenir la chaîne d'équivalences suivante.

$$\forall \ell \in L, \ell \in \text{D-Acc}(\mathcal{A}) \iff \ell \in \text{D-Acc}(\mathcal{A}, \text{Cl\^o}t\text{ure}_{\mathcal{R}}) \iff \ell \in \text{D-Acc}(\mathcal{A}, \text{Approx}_K)$$

Ces équivalences démontrent la propriété (6.3) annoncée page 155 et permettent donc d'établir la correction de l'algorithme 2.

**Exemple 6.7** (Opérateurs  $\text{Cl\^o}t\text{ure}_{\mathcal{R}}$  et  $\text{Approx}_K$ ).

Nous considérons la zone  $Z$  représentée par la surface . Les régions standards, pour la constante 2, sont représentées par des traits pleins. Alors la partie  correspond à  $\text{Approx}_2(Z) \setminus Z$  et la partie  correspond à  $\text{Cl\^o}t\text{ure}_{\mathcal{R}} \setminus \text{Approx}_2(Z)$ .



**Théorème 6.8** (Correction de l'algorithme 2 [Bou04]). *L'algorithme 2 donne une réponse correcte au problème de l'accessibilité d'un état de contrôle pour les classes d'automates temporisés suivantes :*

- la classe des automates temporisés d'Alur et Dill (sans gardes diagonales),





dans l'état noir.

$$\begin{cases} v(x_1) = 0 \\ v(x_2) = d \\ v(x_3) = 2\alpha + 5 \\ v(x_4) = 2\alpha + 5 + d \end{cases}$$

En particulier, cette valuation vérifie la contrainte  $x_2 - x_1 = x_4 - x_3$ , indépendamment des valeurs de  $d$  et  $\alpha$ . Ceci implique donc que la « dernière » transition, entre l'état noir et l'état **Erreur**, n'est jamais franchissable et l'état **Erreur** n'est pas accessible.

Par ailleurs, nous allons montrer que l'algorithme 2 déclare accessible l'état de contrôle **Erreur**. En effet, en notant  $K$  la constante choisie pour l'extrapolation (si  $K$  est définie comme la constante maximale, nous avons ici  $K = 3$ ), il est toujours possible de choisir  $\alpha$  suffisamment grand pour avoir  $2\alpha + 5 > K$ . Fixons donc un tel entier naturel  $\alpha$  et notons  $\rho$  l'exécution correspondante. Définissons enfin  $Z_\rho^e$  (respectivement  $Z_\rho^a$ )<sup>3</sup> comme la zone calculée en itérant l'opérateur **Post** (respectivement l'opérateur  $\text{Approx}_K \circ \text{Post}$  obtenu par composition de  $\text{Approx}_K$  et **Post**) une fois atteint l'état noir.  $Z_\rho^e$  correspond alors à l'ensemble des valuations accessibles le long de cette séquence de transitions, tandis que  $Z_\rho^a$  correspond à l'ensemble des valuations calculées comme étant accessibles par l'algorithme 2 le long de ce même préfixe. Il est facile de démontrer par induction sur la longueur de  $\rho$  que  $Z_\rho^e \subseteq Z_\rho^a$  (en utilisant la croissance de **Post** et le fait que  $\text{Approx}_K$  constitue une sur-approximation). Comme  $\text{Approx}_K(Z_\rho^e)$  est par définition la plus petite (au sens de l'inclusion) zone  $K$ -bornée contenant la zone  $Z_\rho^e$  et puisque  $Z_\rho^a$  est une autre zone  $K$ -bornée contenant  $Z_\rho^e$ , nous obtenons l'inclusion suivante :

$$Z_\rho^e \subseteq \text{Approx}_K(Z_\rho^e) \subseteq Z_\rho^a$$

Nous donnons ci-dessous un système de contraintes pour les zones  $Z_\rho^e$  et  $\text{Approx}_K(Z_\rho^e)$ .

$$Z_\rho^e \left\{ \begin{array}{l} x_1 = 0 \\ 1 \leq x_2 - x_1 \leq 3 \\ 1 \leq x_4 - x_3 \leq 3 \\ x_3 - x_1 = 2\alpha + 5 \\ x_4 - x_2 = 2\alpha + 5 \end{array} \right. \quad \left| \quad \text{Approx}_K(Z_\rho^e) \left\{ \begin{array}{l} x_1 = 0 \\ 1 \leq x_2 - x_1 \leq 3 \\ 1 \leq x_4 - x_3 \leq 3 \\ x_3 - x_1 > K \\ x_4 - x_2 > K \end{array} \right. \right.$$

Notons  $g$  la garde  $x_2 - x_1 \leq 1 \wedge x_4 - x_3 \geq 2$  étiquetant la dernière transition. Nous avons facilement que  $Z_\rho^e \cap \llbracket g \rrbracket = \emptyset$  tandis que  $\text{Approx}_K(Z_\rho^e) \cap \llbracket g \rrbracket \neq \emptyset$ . Ceci implique  $Z_\rho^a \cap \llbracket g \rrbracket \neq \emptyset$  et donc l'algorithme 2 déclare l'état **Erreur** accessible. Remarquons que les inégalités strictes dans la contrainte définissant  $\text{Approx}_K(Z_\rho^e)$  sont dues à l'opérateur d'extrapolation et ne sont pas la cause de l'erreur de l'algorithme.

### 6.3.2 Zones, gardes diagonales et extrapolation

Nous avons vu précédemment que la correction de l'algorithme 2 est obtenue à partir des deux propriétés suivantes :

$$\forall \ell \in L, \ell \in \text{D-Acc}(\mathcal{A}) \iff \ell \in \text{D-Acc}(\mathcal{A}, \text{Clôture}_{\mathcal{R}}) \quad (6.4)$$

$$Z \subseteq \text{Approx}_K(Z) \subseteq \text{Clôture}_{\mathcal{R}}(Z) \quad (6.5)$$

<sup>3</sup>L'exposant  $e$  désigne la zone « exacte » tandis que l'exposant  $a$  désigne la zone « approchée ».

En présence de gardes diagonales, il est nécessaire de modifier l'ensemble de régions considéré afin d'obtenir le premier point. Nous avons introduit dans la sous-section 1.5.3 l'ensemble de régions  $\mathcal{R}_d$  afin de prendre en compte la présence de gardes diagonales. Il est alors facile de vérifier que l'algorithme utilisant l'approximation  $\text{Cl\^o}t\text{ure}_{\mathcal{R}_d}$  est correct vis-à-vis de l'accessibilité d'un état de contrôle. Ceci démontre donc un équivalent de la propriété (6.4) dans le cadre des automates temporisés avec gardes diagonales. Pour tout automate temporisé avec gardes diagonales  $\mathcal{A}$ , nous avons :

$$\forall \ell \in L, \ell \in \text{D-Acc}(\mathcal{A}) \iff \ell \in \text{D-Acc}(\mathcal{A}, \text{Cl\^o}t\text{ure}_{\mathcal{R}_d}) \quad (6.6)$$

Pour le second point en revanche nous allons montrer que l'inclusion n'est plus vérifiée pour ce nouvel ensemble de régions. En effet, l'approximation  $\text{Cl\^o}t\text{ure}_{\mathcal{R}_d}$  est stable vis-à-vis de la satisfaction des gardes diagonales apparaissant dans l'automate (cette implication s'obtient en écrivant  $\llbracket g \rrbracket$  comme une union de régions de  $\mathcal{R}_d$ ) :

$$\forall Z, Z \cap \llbracket g \rrbracket = \emptyset \Rightarrow \text{Cl\^o}t\text{ure}_{\mathcal{R}_d}(Z) \cap \llbracket g \rrbracket = \emptyset$$

Une implication similaire n'est pas vérifiée par  $\text{Approx}_K$ . Ainsi, nous avons montré précédemment que la zone  $Z_\rho^e$  vérifie  $Z_\rho^e \cap \llbracket g \rrbracket = \emptyset$  et  $\text{Approx}_K(Z_\rho^e) \cap \llbracket g \rrbracket \neq \emptyset$  en notant  $g = x_2 - x_1 \leq 1 \wedge x_4 - x_3 \geq 2$  la garde attachée à la dernière transition de l'automate temporisé  $\mathcal{A}$ . Notons  $t$  cette transition, nous obtenons la propriété suivante :

$$\text{Approx}_K(Z_{\rho,t}^e) \not\subseteq \text{Cl\^o}t\text{ure}_{\mathcal{R}_d}(Z_{\rho,t}^e)$$

qui implique que la propriété (6.5) n'est pas vérifiée en présence de contraintes diagonales.

L'algorithme 2 fondé sur l'opérateur d'extrapolation  $\text{Approx}_K$  où  $K$  est la constante maximale apparaissant dans l'automate n'est donc pas correct. Plus généralement, quel que soit le choix de la constante d'extrapolation  $K$ , l'algorithme précédent est encore incorrect puisque dans l'exemple précédent il suffit de choisir pour  $\alpha$  une valeur suffisamment grande par rapport à la valeur donnée à  $K$  pour obtenir la contradiction. Enfin, il est possible d'étendre ce résultat de « non correction » à un ensemble plus général d'opérateurs d'extrapolation. En effet, supposons l'existence d'un opérateur d'extrapolation  $\text{Abs}$  vérifiant les trois propriétés suivantes :

- (i) pour toute zone  $Z$ ,  $\text{Abs}(Z)$  est une zone,
- (ii) pour toute zone  $Z$ ,  $\text{Abs}(Z)$  contient la zone  $Z$ ,
- (iii)  $\text{Abs}$  est à image finie.

Considérons alors l'algorithme 2 dans lequel  $\text{Abs}$  est substitué à  $\text{Approx}_K$ . L'hypothèse (i) permet de s'assurer que cet algorithme ne manipule que des zones, l'hypothèse (ii) permet de garantir sa complétude (*i.e.* une propriété similaire à (6.2)) et enfin l'hypothèse (iii) assure sa terminaison. Comme  $\text{Abs}$  est d'image finie, nous pouvons considérer la constante maximale apparaissant dans les zones éléments de l'image de  $\text{Abs}$ . Notons-la  $M$ . Nous obtenons alors, pour toute zone  $Z$ ,  $Z \subseteq \text{Approx}_M(Z) \subseteq \text{Abs}(Z)$ . Puisque l'algorithme basé sur  $\text{Abs}$  est correct, alors celui basé sur l'opérateur  $\text{Approx}_M$  l'est également, ce qui contredit nos résultats précédents et fournit donc une contradiction.

Ceci implique donc qu'il est vain de chercher à définir un nouvel opérateur d'extrapolation vérifiant les hypothèses (i), (ii) et (iii). Afin de proposer un algorithme correct, plusieurs directions sont alors envisageables que nous étudions dans la section suivante.

## 6.4 Méthodes pour corriger l'algorithme

Nous avons mis en évidence dans la section précédente les problèmes de correction de l'algorithme 2 lorsqu'il est appliqué à des automates temporisés avec gardes diagonales. Nous allons présenter ici différentes pistes qui peuvent sembler envisageables afin de corriger cet algorithme.

### 6.4.1 S'affranchir des gardes diagonales

L'idée la plus naturelle est de chercher à s'appuyer sur la correction du même algorithme, lorsque le modèle ne contient pas de gardes diagonales. Nous avons en effet rappelé que dans ce cas, l'algorithme est correct (théorème 6.8). De plus, nous avons présenté au début de ce document une construction permettant de transformer un automate temporisé avec gardes diagonales en un automate temporisé équivalent sans gardes diagonales (théorème 1.27). Ces deux points regroupés, nous obtenons donc un algorithme correct pour les automates temporisés avec gardes diagonales. Cependant, nous avons remarqué que la construction associée au théorème 1.27 a un coût exponentiel dans le nombre de gardes diagonales. Ainsi, la taille du modèle à analyser va augmenter de façon exponentielle et cette approche ne sera pas efficace en pratique.

Une approche similaire a été proposée dans [BY03]. Intuitivement, le raffinement du modèle vis-à-vis des gardes diagonales, *i.e.* sa transformation en un modèle équivalent sans gardes diagonales, est réalisé au travers de l'opérateur d'extrapolation et non lors d'une phase de pré-calcul. Pour cela, l'opérateur d'extrapolation, au lieu de retourner une zone, renvoie une union finie de zones et chaque zone de l'union est ajoutée individuellement à l'ensemble Successeurs. Plus précisément, le nouvel opérateur d'extrapolation proposé dans ces travaux procède en quatre étapes :

1. Diviser la zone  $Z$  à extrapoler en une union  $Z = \bigsqcup Z_i$  de zones deux à deux disjointes telle que pour tout élément  $Z_i$  de l'union et toute garde diagonale  $g$  du système, l'une des deux inclusions  $Z_i \subseteq \llbracket g \rrbracket$  et  $Z_i \subseteq \llbracket \neg g \rrbracket$  est vérifiée ;
2. Pour chaque zone  $Z_i$ , collecter la liste  $L_i$  des contraintes diagonales  $g$  du système (ou des négations  $\neg g$ ) « satisfaites », *i.e.* telles que  $Z_i \subseteq \llbracket g \rrbracket$  (respectivement telles que  $Z_i \subseteq \llbracket \neg g \rrbracket$ ) ;
3. Extrapoler chaque zone  $Z_i$  à l'aide de l'opérateur  $\text{Approx}_K$ , obtenant ainsi la zone  $W_i$  ;
4. Pour tout  $i$ , raffiner la zone  $W_i$  par les contraintes stockées dans la liste  $L_i$  et retourner la liste des zones obtenues ;

L'algorithme présenté vérifie alors les deux propriétés suivantes :

$$\begin{cases} Z \subseteq \bigcup W_i, \\ \forall i, \forall g, W_i \subseteq \llbracket g \rrbracket \vee W_i \subseteq \llbracket \neg g \rrbracket \end{cases}$$

**Remarque 6.9.** Il est facile de vérifier que les points 2 et 4 de la procédure précédente sont en fait superflus puisque la définition de l'opérateur  $\text{Approx}_K$  assure, si la garde diagonale  $g$  est prise en compte dans le calcul de la constante d'extrapolation  $K$ , que si une zone  $Z$  vérifie  $Z \subseteq \llbracket g \rrbracket$  alors la zone extrapolée vérifie encore cette inclusion :  $\text{Approx}_K(Z) \subseteq \llbracket g \rrbracket$ .  $\lrcorner$

La présentation de l'étape de raffinement du modèle comme une modification de l'opérateur d'extrapolation permet d'éviter le coût du calcul explicite et complet de l'automate équivalent. Cependant, le nombre de zones que l'algorithme va devoir manipuler reste le même et sa complexité est donc elle aussi la même. Globalement, cette seconde méthode souffre donc du même défaut que la première, à savoir une explosion exponentielle (dans le nombre de gardes diagonales présentes) de la complexité de l'algorithme.

### 6.4.2 Détecter les faux-positifs

Une seconde technique consiste à chercher à « vérifier » les réponses retournées par l'algorithme 2. En effet, rappelons que celui-ci calcule une sur-approximation de l'ensemble des états accessibles (propriété 6.2). Les seules erreurs qu'il peut générer sont donc des faux-positifs, *i.e.* déclarer un état accessible alors que celui-ci ne l'est pas. Or il est très facile de modifier légèrement la structure de l'algorithme afin que celui-ci retourne l'exécution symbolique (*i.e.* la séquence finie de transitions discrètes) qui lui a permis d'atteindre cet état. Nous pouvons alors tester la validité de ce contre-exemple en calculant l'ensemble des configurations accessibles le long de cette exécution symbolique afin de vérifier que l'état de contrôle cible est réellement accessible. Ceci peut être réalisé puisque nous avons présenté précédemment comment calculer l'ensemble des successeurs par une transition discrète et puisque l'exécution considérée est finie.

Ceci conduit à la procédure suivante qui intègre une étape de détection des faux-positifs :

1. appliquer l'algorithme 2,
2. si l'état de contrôle cible est déclaré accessible à l'issue d'une exécution symbolique  $\rho$ , tester la validité de cette exécution afin de décider si ce contre-exemple est ou non un faux-positif,
3. si  $\rho$  est un faux-positif, poursuivre l'exécution de l'algorithme 2, sinon arrêter la procédure et déclarer l'état cible accessible.

Cependant, cette procédure peut « manquer » des exécutions réelles menant à l'état de contrôle cible et n'est donc pas complète. En effet, les faux-positifs participent au recouvrement de l'espace d'états accessibles construit par l'algorithme en contribuant à la construction de la liste Visités. Or cette liste permet de limiter l'exploration de certaines transitions discrètes afin de garantir la terminaison de l'algorithme. Une transition menant réellement à l'état cible peut ne pas être explorée à cause d'un faux-positif. Ceci est dû au test d'inclusion permettant de limiter l'exploration de certaines transitions réalisé entre des zones extrapolées car l'inclusion entre deux zones extrapolées n'implique pas l'inclusion entre les deux zones elles-mêmes. Un automate temporisé pour lequel cette situation se produit est représenté sur la figure 6.2 et étudié dans l'exemple 6.10 ci-dessous.

Enfin, l'alternative consistant à retirer de la liste Visités les états déclarés accessibles qui ne le sont en fait pas n'assure plus la terminaison de l'algorithme.

**Exemple 6.10** (Les faux-positifs peuvent dissimuler de vrais contre-exemple). Considérons l'automate temporisé représenté sur la figure 6.2. Dans cet automate, la partie nommée  $\mathcal{A}_0$  dont les contours sont tracés en pointillés représente le sous-ensemble nommé  $\mathcal{A}_0$  de l'automate  $\mathcal{A}$  de la figure 6.1. La constante d'extrapolation, si elle est définie comme la constante maximale apparaissant dans l'automate, vaut 3.

Nous allons démontrer qu'il existe une exécution, passant par la partie « basse » de l'automate, qui permet d'atteindre l'état de contrôle **Erreur** mais qui n'est pas « découverte » par la procédure décrite précédemment. La partie basse de l'automate correspond aux trois états  $r_0$ ,  $r_1$  et  $r_2$ .

Seuls trois paramètres suffisent à décrire toute exécution passant par cette partie basse et atteignant l'état  $q$  : le temps écoulé lors du séjour dans l'état initial, noté  $\delta_1$ , le nombre de fois que la boucle autour de l'état  $r_0$  est franchie, noté  $\beta$  et enfin le délai écoulé dans l'état de contrôle  $r_2$ , noté  $\delta_2$ . La valuation  $v$  atteinte dans l'état  $q$  est alors définie ainsi :

$$v \begin{cases} v(x_1) = 0, \\ v(x_2) = \delta_2, \\ v(x_3) = \beta + 2 + \delta_2, \\ v(x_4) = \delta_1 + \beta + 2 + \delta_2 \end{cases} \quad \text{avec} \quad \begin{cases} 1 \leq \delta_1 \leq 3, \\ \beta \geq 3, \\ 1 \leq \delta_2 \leq 3 \end{cases}$$

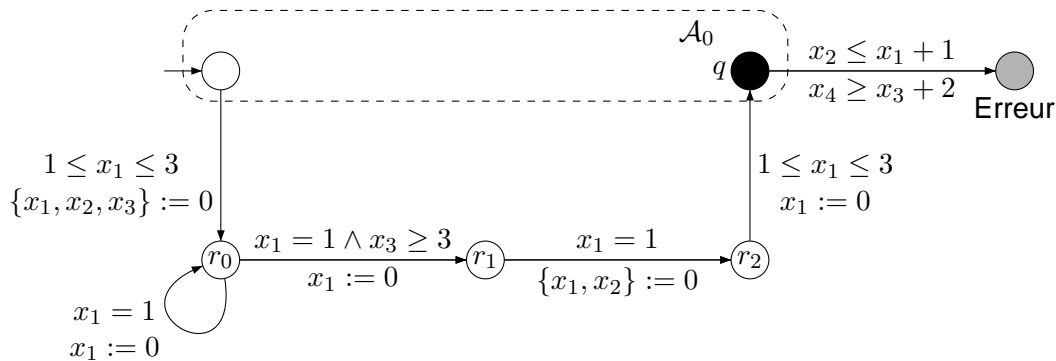


FIG. 6.2 – Un contre-exemple à l'arrêt fondé sur les faux-positifs.

Les deux gardes diagonales sont donc équivalentes pour  $v$  à la contrainte  $\delta_1 \geq 2 \wedge \delta_2 \leq 1$  qui est satisfaisable, ce qui démontre que l'état **Erreur** est accessible.

Montrons que la procédure précédente ne déclare pourtant pas cet état accessible. En effet supposons que celui-ci procède par analyse en largeur de l'espace d'états (s'il procède par analyse en profondeur, il peut choisir d'explorer les transitions de  $\mathcal{A}_0$  avant celles de la partie basse, ce qui mène aux mêmes conclusions). Il atteint alors en premier l'état  $q$  par un chemin passant par l'automate  $\mathcal{A}_0$  qui est de longueur 5. L'état symbolique associé est  $(q, Z_1^a)$  et est ajouté à la liste *Visités*, où la zone  $Z_1^a$  est donnée par :

$$Z_1^a \left\{ \begin{array}{l} 3 < x_3 \\ 1 \leq x_2 \leq 3 \\ x_1 = 0 \\ 1 \leq x_4 - x_3 \leq 3 \\ 3 < x_4 - x_2 \\ 2 \leq x_3 - x_2 \end{array} \right. \quad \left| \quad Z_2^a \left\{ \begin{array}{l} 1 \leq x_2 \leq 3 \\ x_1 = 0 \\ 1 \leq x_4 - x_3 \leq 3 \\ 3 < x_3 - x_2 \end{array} \right.$$

Depuis cet état, l'état **Erreur** est accessible mais le test de validité du contre-exemple détermine qu'il constitue un faux-positif et l'analyse en avant se poursuit. L'algorithme découvre plus tard le chemin menant à l'état  $q$  et passant par la partie basse (atteindre cet état par ce chemin nécessite 7 transitions). L'état symbolique associé est  $(q, Z_2^a)$  où la zone  $Z_2^a$  est donnée ci-dessus. Celle-ci est incluse dans la zone  $Z_1^a$  et l'algorithme détecte donc la présence de l'élément  $(q, Z_1^a)$  dans la liste *Visités* tel que  $Z_2^a \subseteq Z_1^a$ . Il ne calcule donc pas les successeurs de l'état symbolique  $(q, Z_2^a)$  et manque ainsi les exécutions (réelles) atteignant l'état **Erreur**.  $\lrcorner$

À la lumière des difficultés présentées plus haut, une nouvelle proposition est possible. Celle-ci consiste à modifier la procédure décrite plus haut intégrant la détection des faux-positifs en ajoutant une dernière étape fondée sur un calcul en arrière. La nouvelle procédure est la suivante :

- (1) appliquer l'algorithme 2,
- (2) si l'état de contrôle cible est déclaré accessible à l'issue d'une exécution symbolique  $\rho$ , tester la validité de cette exécution afin de décider si ce contre-exemple est ou non un faux-positif,

- (3) si  $\rho$  est un faux-positif, poursuivre l'exécution de l'algorithme 2, sinon arrêter la procédure et déclarer l'état cible accessible.
- (4) à l'issue de l'application de l'algorithme 2, si aucun contre-exemple n'a été détecté, alors déclarer que l'état cible n'est pas accessible. Sinon, seuls des faux-positifs ont été détectés. Dans ce cas, effectuer un calcul en arrière à partir de l'ensemble de l'état de contrôle cible en limitant l'espace exploré à l'espace atteint lors de l'analyse en avant. Si l'état initial est atteint, alors déclarer l'état cible accessible.

Cette nouvelle procédure est correcte. En effet, la première phase de la procédure (étapes (1), (2) et (3)) calcule une sur-approximation de l'espace d'état accessible, et la seconde phase (calcul en arrière) est correcte pour les automates temporisés, même en présence de gardes diagonales. Cependant, nous avons déjà mentionné l'inconvénient d'un algorithme en arrière pour la prise en compte de variables supplémentaires comme des variables entières.

### 6.4.3 Raffinement «à la demande»

Nous proposons maintenant une nouvelle méthode qui constitue un compromis entre les deux types de méthodes précédents. Nous avons présenté d'une part des méthodes qui cherchent à éliminer systématiquement toutes les contraintes diagonales et d'autre part des méthodes fondées sur la détection des faux-positifs obtenue en vérifiant la validité des contre-exemples retournés par l'algorithme. Le défaut des premières propositions est le coût exponentiel de l'élimination des gardes diagonales. Les secondes propositions quant à elles ne permettaient pas d'obtenir un algorithme correct dans le cas général. Notre méthode combine les points positifs des deux techniques, à savoir la correction de la première et l'efficacité de la seconde. Nous utilisons la détection des faux-positifs pour limiter le recours au raffinement du modèle, de sorte à n'effectuer l'opération coûteuse de raffinement que lorsque cela est nécessaire. L'intérêt majeur de cette technique est que notre algorithme ne présente *aucun sur-coût* par rapport à l'algorithme 2 si l'automate temporisé ne produit pas de faux-positif. Comme les automates temporisés avec gardes diagonales donnant lieu à des faux-positifs sont vraisemblablement très peu fréquents, ceci conduit à un algorithme satisfaisant.

Notre méthode que nous présentons est basée sur le paradigme plus général du raffinement itératif fondé sur l'analyse des contre-exemples retournés par l'algorithme. Cette méthodologie, intitulée « CEGAR » pour « Counter-Example Guided Abstraction Refinement », a été introduite par Clarke et al dans [CGJ<sup>+</sup>00] et a été depuis appliquée à de nombreux types de systèmes infinis (programmes à contraintes [HJMS02], systèmes hybrides [ADI03] ...). L'idée générale est d'observer le système à travers une abstraction initialement peu précise et de raffiner ensuite l'abstraction jusqu'à obtenir la solution au problème recherché. Le raffinement de l'abstraction est choisi à partir des faux-positifs obtenus lors de l'analyse du système. Dans notre cadre, les abstractions considérées sont décrites par un ensemble de gardes diagonales du système. Analyser le système avec l'abstraction définie par un ensemble  $\mathcal{D}$  de contraintes diagonales correspond à analyser l'automate en tenant compte de la présence des gardes diagonales éléments de  $\mathcal{D}$ . Deux méthodes peuvent être envisagées pour cette opération : le calcul explicite de l'automate obtenu en appliquant la construction de [BDGP98] pour chaque garde diagonale élément de  $\mathcal{D}$ , ou la prise en compte de  $\mathcal{D}$  dans l'opérateur d'extrapolation selon la méthode définie dans [BY03]. Le choix de l'une ou l'autre de ces deux propositions n'a pas d'importance pour la correction de notre méthode, nous parlerons simplement d'analyse de  $\mathcal{A}$  en tenant compte de la présence de  $\mathcal{D}$ , et nous noterons « Analyse de  $(\mathcal{A}, \mathcal{D})$  ». Enfin, raffiner l'abstraction revient à augmenter l'ensemble  $\mathcal{D}$ .

Le test de la validité de l'exécution a été présenté dans la sous-section 6.4.2. La sélection d'une garde diagonale  $g$  responsable du faux-positif constitue la partie délicate de cette méthode. Ce problème est

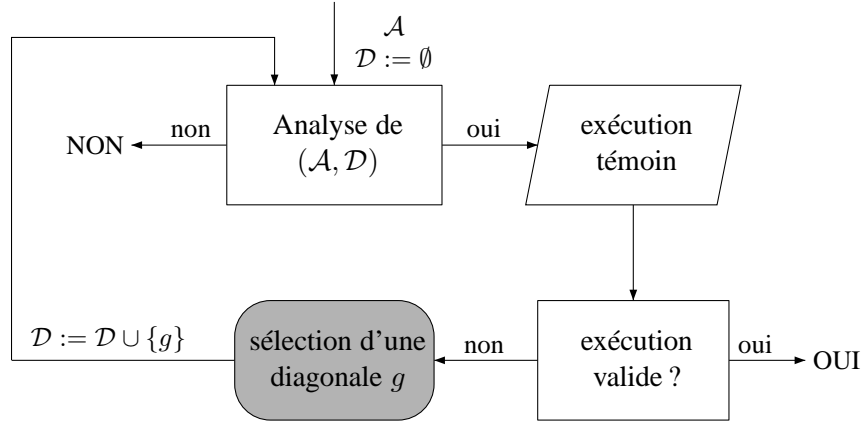


FIG. 6.3 – Méthode fondée sur le raffinement itératif

l'objet de la prochaine section. D'autre part, la terminaison de cette procédure est assurée dès lors que la méthode permettant de sélectionner une garde diagonale  $g$  responsable du faux-positif choisit toujours une nouvelle garde diagonale. En effet, l'ensemble des gardes diagonales est fini et dans le pire cas, au bout d'un nombre fini d'itérations, l'analyse de  $\mathcal{A}$  est donc réalisée en considérant pour  $\mathcal{D}$  l'ensemble des gardes diagonales de  $\mathcal{A}$ . Dans ce cas, l'analyse est correcte, d'après la correction des méthodes présentées dans la sous-section 6.4.1 et il n'est donc plus possible d'obtenir un faux-positif. Enfin, la procédure générale est correcte puisque la réponse « NON » est toujours correcte et la réponse « OUI » n'est retournée qu'après avoir testé le contre-exemple correspondant.

## 6.5 Analyse des contre-exemples

La méthode que nous présentons est en fait couplée avec le test de validité de l'exécution : nous fusionnons donc ces deux étapes et le coût de la procédure que nous obtenons n'est pas supérieur au coût du test de validité de l'exécution.

### 6.5.1 Définitions préliminaires

Considérons un automate temporisé  $\mathcal{A} = (\Sigma, X, L, T, \ell_0, L_f, L_r)$ , notons  $n$  son nombre d'horloges et  $K$  sa constante maximale. Notons  $\text{Diag}(\mathcal{A})$  l'ensemble des gardes diagonales apparaissant dans  $\mathcal{A}$ .

Une *trace*  $\rho$  dans  $\mathcal{A}$  est une exécution symbolique, *i.e.* une séquence finie de transitions consécutives de  $\mathcal{A}$ . Étant donnée une trace  $\rho$  dans  $\mathcal{A}$ , nous associons à  $\rho$  deux zones<sup>4</sup> définies par induction sur la longueur de  $\rho$  :

$$\begin{cases} Z_\rho^e = Z_\rho^a = \overrightarrow{\{0\}} & \text{si } \rho = \varepsilon \\ Z_\rho^e = \text{Post}(Z_{\rho'}^e, t) \text{ et } Z_\rho^a = \text{Approx}_K(\text{Post}(Z_{\rho'}^a, t)) & \text{si } \rho = \rho'.t \end{cases}$$

Intuitivement,  $Z_\rho^e$  correspond à l'ensemble des valuations accessibles à l'issue de l'exécution symbolique  $\rho$  tandis que  $Z_\rho^a$  correspond à l'ensemble des valuations calculées comme étant accessibles par l'algorithme 2 à l'issue de l'exécution symbolique  $\rho$ .

<sup>4</sup>À nouveau, l'exposant  $e$  désigne la zone « exacte » tandis que l'exposant  $a$  désigne la zone « approchée ».

Étant donné un état de contrôle  $\ell_c \in L$ , nous dirons que  $\rho$  est un *témoin de l'accessibilité de  $\ell_c$*  si et seulement si  $\rho$  est une trace menant de l'état de contrôle initial à l'état de contrôle  $\ell_c$  telle que  $Z_\rho^a \neq \emptyset$ . Si de plus  $Z_\rho^e \neq \emptyset$  alors  $\rho$  est un *vrai témoin* tandis que sinon ( $Z_\rho^e = \emptyset$ ), nous dirons que  $\rho$  est un *faux témoin*.

Dans la suite, nous présentons un algorithme qui, étant donné un faux témoin  $\rho$  de l'accessibilité d'un état  $\ell_c$ , calcule un ensemble de gardes diagonales  $G \subseteq \text{Diag}(\mathcal{A})$  tel que les deux propriétés suivantes sont satisfaites :

(i)  $G \neq \emptyset$ ,

(ii) lors de l'analyse de  $(\mathcal{A}, G)$ , la trace  $\rho$  n'est plus un témoin de l'accessibilité de  $\ell_c$ .

La propriété (i) permet d'assurer la terminaison de la méthode fondée sur le raffinement itératif. La propriété (ii) énonce une notion de progrès : une fois les éléments de  $G$  éliminés, l'algorithme 2 ne déclare plus la trace  $\rho$  comme témoin et elle ne sera donc pas considérée plusieurs fois. Remarquons que nous avons autorisé ici  $G$  à être un ensemble de gardes diagonales et non un singleton. Plusieurs « politiques » de choix sont alors possibles : raffiner directement selon  $G$  et s'assurer de l'élimination de  $\rho$  ou raffiner pas à pas en ajoutant les éléments de  $G$  à l'ensemble des gardes diagonales à éliminer un à un.

Naturellement, l'ensemble  $\text{Diag}(\mathcal{A})$  satisfait les deux conditions (i) et (ii). Cependant, ceci constitue le choix le plus désavantageux en termes de complexité et en pratique notre algorithme peut sélectionner des ensembles nettement plus petits que l'ensemble  $\text{Diag}(\mathcal{A})$ .

D'autre part, afin de simplifier notre étude, nous décomposons les franchissements des transitions en pas élémentaires. Ainsi, dans le calcul de la zone exacte  $Z_\rho^e$ , seuls trois types d'opérations peuvent intervenir : le calcul de l'intersection avec une garde atomique<sup>5</sup>, la remise à zéro d'une horloge et le calcul des successeurs temporels. Nous obtenons donc la décomposition suivante :

$$\ell \xrightarrow{\bigwedge_{j=1}^p g_j, \{x_1, \dots, x_m\}} \ell' \quad \text{est transformé en} \quad \ell \xrightarrow{\bigcap g_1} \dots \xrightarrow{\bigcap g_p} \xrightarrow{x_1 \leftarrow 0} \dots \xrightarrow{x_k \leftarrow 0} \xrightarrow{Fut.} \ell'$$

Dans la suite, nous écrivons les traces  $\rho$  comme des séquences  $\alpha_1 \dots \alpha_q$  de tels pas élémentaires. La seule opération qui peut mener à une zone vide est l'intersection avec une garde atomique. Étant donné un témoin  $\rho = \alpha_1 \dots \alpha_q$ , nous définissons pour tout indice  $j \in \{1, \dots, q\}$  le préfixe  $\rho_j$  de  $\rho$  par  $\rho_j = \alpha_1 \dots \alpha_j$ . L'*indice d'erreur* est alors obtenu par  $i = \min\{j \in \{1, \dots, q\} \mid Z_{\rho_j}^e = \emptyset\}$ . Cet indice existe si et seulement si  $\rho$  est un faux témoin. Si c'est le cas, alors l'opération  $\alpha_i$  est une intersection avec une garde atomique et cette garde atomique est appelée la *garde atomique d'erreur*.

## 6.5.2 Quelques contre-exemples

Les intuitions naturelles suivantes se révèlent être erronées :

- (a) si la trace  $\rho$  ne contient pas de gardes diagonales, alors l'approximation  $Z_\rho^a$  est correcte, i.e.  $Z_\rho^a = \text{Approx}_K(Z_\rho^e)$ .
- (b) la garde atomique d'erreur est nécessairement une garde diagonale,
- (c) si la garde atomique d'erreur est une garde diagonale, celle-ci est responsable,

Afin de contredire chacune de ces trois intuitions, nous donnons un contre-exemple à chacune d'entre elles. Pour l'intuition (a), considérons l'automate temporisé représenté sur la figure 6.4, obtenu comme une simple transformation de l'automate de la figure 6.1. La zone  $Z_\rho^a$  atteinte à l'issue de

<sup>5</sup>Une garde atomique est une garde de la forme  $x \sim c$  ou  $x - y \sim c$  avec  $\sim \in \{<, \leq, =, \geq, >\}$ .



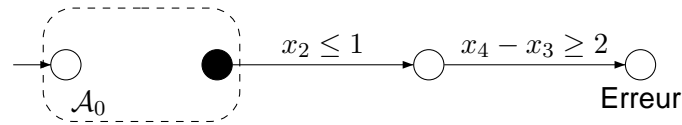


FIG. 6.4 – Des gardes diagonales provoquent des approximations.

la trace se finissant par  $x_2 \leq 1$  possède un coefficient  $(4, 3)$  erroné. Celui-ci vaut en effet  $(\leq, 3)$  au lieu de  $(\leq, 1)$  puisque la transition est franchissable dans le calcul approché mais pas dans le calcul exact.

L'automate représenté sur la figure 6.5 contredit l'intuition (b). La dernière transition porte la garde non diagonale  $x_2 \leq 1$  qui sera la garde atomique d'erreur dans ce cas. En effet, il suffit de constater que si la garde  $x_2 \leq 1$  est vérifiée, alors la garde  $x_2 - x_1 \leq 1$  l'est aussi.

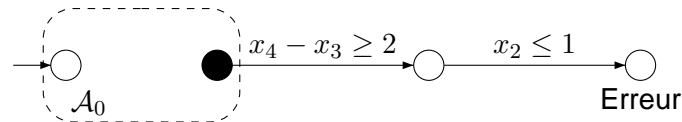


FIG. 6.5 – La garde atomique d'erreur peut ne pas être diagonale.

Pour l'intuition (c), l'automate représenté sur la figure 6.6 fournit un contre-exemple. La construction consiste à utiliser une horloge supplémentaire  $x_5$  qui intervient dans la garde atomique associée à l'indice d'erreur. Cette horloge n'est pas utilisée dans la première partie de l'automate (*i.e.* la partie  $\mathcal{A}_0$ ) excepté sur la dernière transition où elle est remise à zéro. Elle est ensuite utilisée à la place de l'horloge  $x_1$  sur la dernière transition, provoquant l'erreur de l'algorithme puisque  $x_5$  et  $x_1$  ont été remises à zéro simultanément. Pourtant il est facile de vérifier que le raffinement de l'automate vis-à-vis de la garde diagonale  $x_2 - x_5 \leq 1$  ne suffit pas à éliminer le faux témoin.

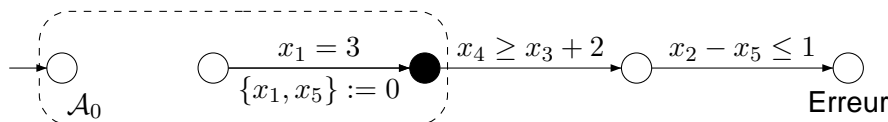


FIG. 6.6 – La garde atomique d'erreur, même diagonale, peut ne pas être responsable.

### 6.5.3 Propagation des contraintes diagonales

La garde diagonale responsable d'un faux témoin peut intervenir à différents endroits dans ce faux témoin. En particulier, elle peut intervenir arbitrairement loin avant la garde atomique d'erreur, qui elle-même peut ne pas être diagonale ! La raison sous-jacente est que dans l'approximation  $Z_\rho^a$  calculée par l'algorithme de la zone exacte  $Z_\rho^e$ , des erreurs peuvent exister depuis arbitrairement longtemps avant d'atteindre la garde atomique d'erreur. Afin de pouvoir, une fois cette garde atteinte, obtenir des informations sur les gardes diagonales responsables de ces erreurs, nous allons mémoriser tout au long du calcul de la zone  $Z_\rho^e$ , et pour chacun de ses coefficients, quelles sont les gardes diagonales qui ont été utilisées. Il est important de remarquer que ceci ne revient pas à considérer l'ensemble des

gardes diagonales présentes sur la trace témoin avant la garde atomique d'erreur. Nous allons en effet mémoriser uniquement les gardes diagonales réellement mises en jeu dans le calcul de chacun des coefficients

Comme nous utilisons un algorithme de plus court chemin lors de la normalisation de la représentation des zones, les valeurs d'une entrée peuvent dépendre des valeurs des autres entrées, ce qui explique pourquoi nous devons calculer les dépendances de chacun des coefficients de la matrice à différences bornées représentant la zone.

Par exemple, si la zone courante est  $x_2 \leq 5$  et si la transition suivante dans la trace est  $\xrightarrow{g, \{x_2\}}$  où  $g$  dénote la contrainte  $x_1 - x_2 \leq 3$ , alors la zone suivante vaut  $x_1 \leq 8 \wedge x_2 = 0 \wedge x_1 - x_2 \leq 8$ . Nous mémorisons alors que les contraintes  $x_1 \leq 8$  et  $x_1 - x_2 \leq 8$  dépendent de la contrainte  $g$  parce qu'elles sont obtenues à partir de l'intersection de  $g$  avec la contrainte  $x_2 \leq 5$ . Ce n'est pas le cas de la contrainte  $x_2 = 0$  qui ne dépend que de la remise à zéro de  $x_2$ . Afin de mémoriser de telles dépendances, nous devons modifier la structure de données des matrices à différences bornées. Pour cela, nous définissons les matrices à différences bornées étendues. Notons que dans cette définition, les coefficients des matrices à différences bornées ne sont pas des paires  $(\prec, m)$  appartenant à l'ensemble  $\{\prec, \leq\} \times \mathbb{Z}$  comme dans la définition 6.5 mais seulement des éléments de  $\mathbb{Z}$ . Cette définition est correcte uniquement pour les automates temporisés n'utilisant que des contraintes larges. Cette simplification nous permet d'obtenir des notations plus simples pour la lecture du document. Cependant, notre méthode s'étend trivialement au modèle des matrices à différences bornées tel qu'il a été introduit dans la définition 6.5.

**Définition 6.11** (Matrices à différences bornées étendues). *Une matrice à différences bornées étendue est une paire  $(M, \mathcal{S})$  de matrices carrées de dimension  $n + 1$  où  $M$  est une matrice à différences bornées et  $\mathcal{S}$  est une matrice dont les coefficients sont des ensembles non vides de sous-ensembles de  $\text{Diag}(\mathcal{A})$ .*

Nous considérons une matrice à différences bornées étendue  $(M, \mathcal{S})$  avec  $M = (m_{i,j})_{0 \leq i,j \leq n}$  et  $\mathcal{S} = (\mathcal{S}_{i,j})_{0 \leq i,j \leq n}$ . Cette matrice à différences bornées étendue représente la même zone que la matrice à différences bornées  $M$  et l'ensemble  $\mathcal{S}_{i,j}$  contient intuitivement l'ensemble des gardes diagonales desquelles le coefficient  $m_{i,j}$  dépend. Plus précisément, lors du calcul de la contrainte  $m_{i,j}$  par l'algorithme de normalisation (algorithme de Floyd), plusieurs chemins peuvent être à l'origine de la valeur minimale. À chacun de ces chemins correspond un ensemble de gardes diagonales mises en jeu, c'est pourquoi le coefficient  $\mathcal{S}_{i,j}$  de la matrice  $\mathcal{S}$  contient l'ensemble de ces ensembles de gardes diagonales. Le type du coefficient  $\mathcal{S}_{i,j}$  est donc un ensemble non vide de sous-ensembles de  $\text{Diag}(\mathcal{A})$ . Chaque ensemble  $G \in \mathcal{S}_{i,j}$  est un candidat possible pour l'étape de raffinement si le coefficient  $m_{i,j}$  est le coefficient responsable de la garde atomique d'erreur. Nous mémorisons tous les sous-ensembles possibles afin de pouvoir choisir le sous-ensemble minimal à l'issue de l'analyse du faux témoin.

**Opérations sur les matrices à différences bornées étendues.** Étant donnée une contrainte  $g$ , nous définissons l'ensemble  $\text{Ens}(g)$  ainsi :

$$\text{Ens}(g) = \begin{cases} \{\{g\}\} & \text{si } g \text{ est une garde diagonale,} \\ \{\emptyset\} & \text{sinon.} \end{cases}$$

Nous définissons également les deux opérations suivantes qui permettent de manipuler les ensembles non-vides d'ensembles :

$$(i) \quad \mathcal{S}_1 \vee \mathcal{S}_2 = \{G \mid G \in \mathcal{S}_1 \cup \mathcal{S}_2\}$$

$$(ii) \mathcal{S}_1 \wedge \mathcal{S}_2 = \{G_1 \cup G_2 \mid G_1 \in \mathcal{S}_1 \text{ et } G_2 \in \mathcal{S}_2\}$$

Nous pouvons à présent étendre les opérations classiques sur les matrices à différences bornées au cadre des matrices à différences bornées étendues. Nous considérons donc deux matrices à différences bornées étendues  $(M, \mathcal{S})$  et  $(M', \mathcal{S}')$  avec  $M = (m_{i,j})_{i,j=0\dots n}$ ,  $\mathcal{S} = (\mathcal{S}_{i,j})_{i,j=0\dots n}$ ,  $M' = (m'_{i,j})_{i,j=0\dots n}$  et  $\mathcal{S}' = (\mathcal{S}'_{i,j})_{i,j=0\dots n}$ . Nous supposons de plus que la matrice à différences bornées  $M$  est sous forme normale.

*Futur.*  $(M', \mathcal{S}') = \overrightarrow{(M, \mathcal{S})}$  dès lors que :

$$(m'_{i,j}, \mathcal{S}'_{i,j}) = \begin{cases} (\infty, \{\emptyset\}) & \text{si } j = 0 \\ (m_{i,j}, \mathcal{S}_{i,j}) & \text{sinon.} \end{cases}$$

*Remise à zéro de l'horloge  $x_k$ .*  $(M', \mathcal{S}') = [x_k \leftarrow 0](M, \mathcal{S})$  dès lors que

$$(m'_{i,j}, \mathcal{S}'_{i,j}) = \begin{cases} (0, \{\emptyset\}) & \text{si } i, j \in \{0, k\} \\ (m_{i,0}, \mathcal{S}_{i,0}) & \text{si } j = k, \\ (m_{0,j}, \mathcal{S}_{0,j}) & \text{si } i = k, \\ (m_{i,j}, \mathcal{S}_{i,j}) & \text{sinon} \end{cases}$$

*Intersection avec  $g = (x_k - x_l \leq c)$ .*  $(M', \mathcal{S}') = \text{Inter}((M, \mathcal{S}), g)$  dès lors que :

$$(m'_{i,j}, \mathcal{S}'_{i,j}) = \begin{cases} (m_{i,j}, \mathcal{S}_{i,j}) & \text{si } m_{i,j} < m \\ (m_{i,j}, \mathcal{S}_{i,j} \vee \overline{\mathcal{S}}) & \text{si } m_{i,j} = m \\ (m, \overline{\mathcal{S}}) & \text{si } m_{i,j} > m \end{cases}$$

$$\text{où } m = m_{i,k} + c + m_{l,j} \text{ et } \overline{\mathcal{S}} = \mathcal{S}_{i,k} \wedge \text{Ens}(g) \wedge \mathcal{S}_{l,j}$$

Remarquons que l'opération d'intersection inclut la phase de normalisation qui renforce chaque coefficient par rapport à la contrainte  $g$ . Ainsi, la matrice à différences bornées  $M'$  qui résulte de chacune de ces trois opérations est sous forme normale.

À l'aide de la décomposition des traces en pas élémentaires, ces trois opérations permettent de calculer la zone  $Z_\rho^e$  pour toute trace  $\rho$ . Étant donnée une trace  $\rho$ , nous définissons naturellement par induction sur  $\rho$  la notation  $(M_\rho, \mathcal{S}_\rho)$  comme la matrice à différences bornées étendue obtenue à l'issue de l'application de l'opération correspondante, pour chacune des opérations élémentaires constituant la trace  $\rho$ . Initialement, nous définissons  $M_\varepsilon$  comme la matrice à différences bornées représentant la zone  $\{\overline{0}\}$  et  $\mathcal{S}_\varepsilon$  comme une matrice dont tous les coefficients valent  $\{\emptyset\}$ . En particulier, pour toute trace  $\rho$ , nous avons alors  $Z_\rho^e = \llbracket M_\rho \rrbracket$ . De plus, afin de faciliter les notations, nous écrivons  $M_\rho(i, j)$  (respectivement  $\mathcal{S}_\rho(i, j)$ ) pour représenter le coefficient  $(i, j)$  de la matrice  $M_\rho$  (respectivement  $\mathcal{S}_\rho$ ).

#### 6.5.4 Présentation et correction de la méthode

**Présentation de l'algorithme.** Notre algorithme pour l'analyse des témoins et pour la sélection des gardes diagonales responsables des faux témoins est présenté comme l'algorithme 4. Nous le nommons Choisir\_gardes. Sa structure est simple, au lieu de calculer l'ensemble des successeurs le long du témoin  $\rho$  à l'aide de matrices à différences bornées, nous effectuons ce calcul à l'aide de matrices à différences bornées étendues. À l'issue de la  $i$ -ème étape de cette itération, la matrice à différences bornées étendue  $(M', \mathcal{S}')$  est notée  $(M_{\rho_i}, \mathcal{S}_{\rho_i})$  en écrivant  $\rho_i = \alpha_1 \dots \alpha_i$ . Deux cas peuvent survenir :  $\rho$  est ou bien un vrai témoin ou bien un faux témoin. Dans le premier cas, nous avons par définition  $Z_\rho^e \neq \emptyset$ . D'après la propriété énoncée précédemment établissant que  $\llbracket M_{\rho'} \rrbracket = Z_{\rho'}^e$  pour tout préfixe  $\rho'$  de  $\rho$ , la zone atteinte à l'issue de  $\rho$  n'est pas vide :  $\llbracket M_{\rho_p} \rrbracket = Z_{\rho_p}^e \neq \emptyset$ . Ainsi, le test

**Algorithme 4** Analyse des témoins et sélection des gardes diagonales – Choisir\_gardes**Entrée :**  $\rho = \alpha_1 \dots \alpha_p$  un témoin de l'accessibilité de  $\ell_c$  dans  $\mathcal{A}$ **Sortie :** Réponse à “ $\rho$  est-il un vrai ou un faux témoin ?” Si c'est un faux témoin, sélection d'un ensemble de gardes diagonales responsables.

```

1:  $(M, \mathcal{S}) := (M_\epsilon, \mathcal{S}_\epsilon), (M', \mathcal{S}') := (M_\epsilon, \mathcal{S}_\epsilon);$  /* Initialisation */
2:  $i := 0;$ 
3: Répéter
4:    $i := i + 1;$ 
5:    $(M, \mathcal{S}) := (M', \mathcal{S}');$ 
6:   Si  $\alpha_i$  est l'opération d'intersection avec la garde  $g_i$  Alors
7:      $(M', \mathcal{S}') := \text{Inter}((M, \mathcal{S}), g_i);$ 
8:   Fin Si
9:   Si  $\alpha_i$  est l'opération de remise à zéro de l'horloge  $x$  Alors
10:     $(M', \mathcal{S}') := [x \leftarrow 0](M, \mathcal{S});$ 
11:   Fin Si
12:   Si  $\alpha_i$  est l'opération d'écoulement du temps Alors
13:     $(M', \mathcal{S}') := \overrightarrow{(M, \mathcal{S})};$ 
14:   Fin Si
15: Jusqu'à  $\llbracket M' \rrbracket = \emptyset \vee i = p$ 
16: Si  $\llbracket M' \rrbracket \neq \emptyset$  Alors /* Cas d'un vrai témoin */
17:   Retourner «  $\rho$  est un vrai témoin »
18: Sinon /* Cas d'un faux témoin */
19:   Écrire  $g_i = x_k - x_l \leq c$  /*  $k$  ou  $l$  peut être nul */
20:    $\mathcal{S}_{\text{resp}} := \mathcal{S}(l, k) \wedge \text{Ens}(g_i);$ 
21:   Retourner «  $\rho$  est un faux témoin, causé par  $\mathcal{S}_{\text{resp}}$  »
22: Fin Si

```

de la ligne 16 est vérifié et l'algorithme déclare le témoin comme un vrai témoin. Dans le cas contraire, *i.e.* si  $\rho$  est un faux témoin,  $\rho$  admet un indice d'erreur  $i \in \{1, \dots, p\}$  défini page 168 comme l'indice minimal à partir duquel la zone correspondante  $Z_{\rho_i}^e$  est vide. La boucle « **Répéter** » est interrompue précisément au niveau de cet indice grâce à la première condition du test d'arrêt de la ligne 15. Les matrices à différences bornées correspondantes vérifient alors les égalités suivantes (l'indice d'erreur est  $i + 1$ ) :

$$\begin{cases} \llbracket M_{\rho_i} \rrbracket \neq \emptyset \\ \llbracket M_{\rho_i, \alpha_i} \rrbracket = \llbracket \text{Post}(M_{\rho_i}, \alpha_i) \rrbracket = \emptyset \end{cases}$$

L'opération  $\alpha_i$  est nécessairement l'intersection avec une certaine garde  $g$  que nous écrivons  $x_k - x_l \leq c$  (ligne 19), où un des deux indices  $k$  et  $l$  peut éventuellement être nul (la garde  $g$  n'est pas nécessairement une garde diagonale). Les propriétés précédentes impliquent alors  $M_{\rho_i}(l, k) + c < 0$  (sinon la zone  $\llbracket M_{\rho_i, \alpha_i} \rrbracket$  ne serait pas vide). Au contraire, le coefficient correspondant dans la zone  $Z_{\rho_i}^a$  calculée par l'algorithme 2 vérifie l'inégalité contraire puisque la zone  $Z_{\rho_i, \alpha_i}^a$  n'est pas vide. Intuitivement, ceci implique donc que le coefficient  $(l, k)$  a été « mal évalué » par l'algorithme 2 du fait des extrapolations. C'est pourquoi l'algorithme 4 retourne à la ligne 20 l'ensemble  $\mathcal{S}_{\rho_i}(l, k)$  et ajoute la garde  $g$  si celle-ci est une garde diagonale. Ajouter la garde  $g$  peut en effet être nécessaire : considérons l'automate représenté sur la figure 6.5 dans lequel les deux dernières transitions sont inversées. L'erreur intervient lors de la dernière transition et aucune des transitions précédentes n'est

étiquetée par une garde diagonale.

**Correction de l'algorithme.** Si  $\rho$  est une trace, et  $G$  un sous-ensemble de  $\text{Diag}(\mathcal{A})$ , nous notons  $\rho[G \leftarrow \text{tt}]$  la trace obtenue en remplaçant toutes les transitions étiquetées par une contrainte  $g \in G$  par la transition étiquetée par la contrainte  $\text{tt}$ . Intuitivement, ce premier lemme établit que les gardes diagonales qui n'ont pas été sélectionnées par notre algorithme ne sont pas impliquées dans le calcul du coefficient correspondant.

**Lemme 6.12.** Soit  $\mathcal{A}$  un automate temporisé et  $\rho$  une trace dans  $\mathcal{A}$ . Considérons une paire  $(i, j)$  et un ensemble de gardes diagonales  $G$  inclus dans  $\text{Diag}(\mathcal{A})$ .

$$\text{Si } \left( \exists G_0 \in \mathcal{S}_\rho(i, j) \text{ t.q. } G_0 \cap G = \emptyset \right) \text{ alors } M_{\rho[G \leftarrow \text{tt}]}(i, j) = M_\rho(i, j).$$

Avant de démontrer le lemme 6.12, nous établissons le lemme suivant :

**Lemme 6.13.** Soit  $\rho$  une trace dans un automate temporisé  $\mathcal{A}$  et  $G$  un ensemble de gardes diagonales. Alors, pour toute paire  $(i, j)$ ,  $M_\rho(i, j) \leq M_{\rho[G \leftarrow \text{tt}]}(i, j)$ .

**Preuve.** Par définition, la trace  $\rho[G \leftarrow \text{tt}]$  est moins contraignante que la trace  $\rho$ , puisque nous avons relâché certaines contraintes de franchissement. Nous avons donc l'inclusion suivante en termes de zones :  $Z_\rho^e \subseteq Z_{\rho[G \leftarrow \text{tt}]}^e$ . Ceci implique l'inégalité correspondante sur les coefficients des matrices à différences bornées associées, puisque celles-ci sont en forme normale.  $\square$

**Preuve du lemme 6.12.** La preuve du lemme 6.12 est réalisée par induction sur la longueur de  $\rho$  : considérons donc une trace  $\rho_1$  et étendons là en une trace  $\rho_2$  à l'aide d'une opération élémentaire  $\alpha : \rho_2 = \rho_1 \cdot \alpha$ . Supposons de plus que la propriété énoncée dans le lemme est correcte pour  $\rho_1$  et démontrons qu'elle est encore valable pour  $\rho_2$  en distinguant les différents cas possibles pour  $\alpha$ . Fixons deux indices  $i$  et  $j$  et un ensemble de gardes diagonales  $G_0$  tel qu'il existe  $G \in \mathcal{S}_{\rho_2}(i, j)$  vérifiant  $G_0 \cap G = \emptyset$ . Nous fixons un tel ensemble  $G$ .

– **Cas où  $\alpha$  est l'intersection avec la contrainte** ( $\gamma : x_k - x_l \leq c$ ).

Rappelons d'abord la définition du calcul de l'intersection avec la contrainte  $\gamma$  :

$$\mathcal{S}_{\rho_2}(i, j) = \begin{cases} \mathcal{S}_{\rho_1}(i, j) & \text{if } M_{\rho_1}(i, j) < m \\ \mathcal{S}_{\rho_1}(i, j) \vee \mathcal{S}_0 & \text{if } M_{\rho_1}(i, j) = m \\ \mathcal{S}_0 & \text{if } M_{\rho_1}(i, j) > m \end{cases}$$

où  $m = M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j)$  et  $\mathcal{S}_0 = \mathcal{S}_{\rho_1}(i, k) \wedge \text{Ens}(\gamma) \wedge \mathcal{S}_{\rho_1}(l, j)$ . Ainsi, comme  $G \in \mathcal{S}_{\rho_2}(i, j)$ , nous avons  $G \in \mathcal{S}_{\rho_1}(i, j)$  ou  $G \in \mathcal{S}_0$  (ou encore les deux) et nous pouvons donc dans tous les cas appliquer l'hypothèse d'induction à au moins un des deux ensembles  $\mathcal{S}_{\rho_1}(i, j)$  et  $\mathcal{S}_0$ .

– **(a)** Nous appliquons l'hypothèse d'induction à l'ensemble  $\mathcal{S}_{\rho_1}(i, j)$ .

Par définition de l'intersection, ceci ne peut se produire que si  $M_{\rho_1}(i, j) \leq M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j)$ . Nous supposons donc cela à présent. Nous avons alors  $M_{\rho_2}(i, j) = M_{\rho_1}(i, j)$ . Par hypothèse d'induction, nous obtenons  $M_{\rho_1}(i, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j)$ . Deux cas peuvent alors se produire pour  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j)$  :

- (i) ou bien  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j) \leq M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j)$
- (ii) ou bien  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j) > M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j)$ .

Dans le premier cas, nous obtenons que  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j) = M_{\rho_1}(i, j) = M_{\rho_2}(i, j)$  ce qui démontre le résultat.

Nous allons démontrer que le second cas ne peut pas se produire. Par définition de l'intersection, nous avons dans ce cas  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j)$ .

Par hypothèse d'induction, nous avons  $M_{\rho_1}(i, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j)$ . Par (ii), nous avons également  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j) > M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j)$ . Par le lemme 6.13, nous avons  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) \geq M_{\rho_1}(i, k)$  et  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j) \geq M_{\rho_1}(l, j)$ , alors  $M_{\rho_1}(i, j) > M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j)$  ce qui contredit l'hypothèse de départ.

- (b) Nous appliquons l'hypothèse d'induction à l'ensemble  $\mathcal{S}_0 = \mathcal{S}_{\rho_1}(i, k) \wedge \text{Ens}(\gamma) \wedge \mathcal{S}_{\rho_1}(l, j)$ . Par définition de l'intersection, ceci ne peut survenir que si  $M_{\rho_1}(i, j) \geq M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j)$ . C'est ce que nous supposons désormais. Alors nous avons  $M_{\rho_2}(i, j) = M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j)$ . Puisque  $G \in \mathcal{S}_0$ , nous avons  $G = G_1 \cup \{\gamma\} \cup G_2$  (ou  $G = G_1 \cup G_2$  selon que  $\gamma$  est ou non une garde diagonale), où  $G_1 \in \mathcal{S}_{\rho_1}(i, k)$  et  $G_2 \in \mathcal{S}_{\rho_1}(l, j)$ . Alors nous pouvons appliquer l'hypothèse d'induction aux deux ensembles  $\mathcal{S}_{\rho_1}(i, k)$  et  $\mathcal{S}_{\rho_1}(l, j)$  et nous obtenons finalement  $M_{\rho_1}(i, k) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k)$  et  $M_{\rho_1}(l, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j)$ . Deux cas peuvent survenir pour  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j)$  :

- (i) ou bien  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j) \geq M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j)$
- (ii) ou bien  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j) < M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j)$ .

Dans le cas (i), nous obtenons  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j) = M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j) = M_{\rho_2}(i, j)$  ce qui démontre le résultat.

Nous allons démontrer que le second cas ne peut pas survenir. Par définition de l'intersection, nous avons dans ce cas  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j)$ . D'après (ii), nous avons  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j) < M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j)$ . L'hypothèse d'induction implique aussi  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, k) = M_{\rho_1}(i, k)$  et  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(l, j) = M_{\rho_1}(l, j)$  et nous avons donc :  $M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j) < M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j)$ . Mais l'hypothèse initiale donne  $M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j) = M_{\rho_2}(i, j)$ . Le lemme 6.13 fournit donc  $M_{\rho_2}(i, j) \leq M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j)$  ce qui donne une contradiction.

– **Cas où  $\alpha$  est l'opération de futur.**

Si  $j \neq 0$ , alors par définition de  $\overline{(M, \mathcal{S})}$ , nous avons  $\mathcal{S}_{\rho_2}(i, j) = \mathcal{S}_{\rho_1}(i, j)$ . Nous pouvons ainsi appliquer l'hypothèse d'induction à  $\mathcal{S}_{\rho_1}(i, j)$  ce qui nous donne  $M_{\rho_1}(i, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j)$ . De plus, nous avons également  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j) = M_{\rho_1[G_0 \leftarrow \text{tt}]}(i, j)$  et  $M_{\rho_2}(i, j) = M_{\rho_1}(i, j)$  car  $j \neq 0$ . Nous obtenons ainsi le résultat escompté.

Si  $j = 0$ , alors à la fois  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j)$  et  $M_{\rho_2}(i, j)$  sont égaux à  $+\infty$ .

– **Cas où  $\alpha$  est la remise à zéro de l'horloge  $x_k$ .**

Dans le cas où  $(i, j) \in \{(k, k), (0, k), (k, 0)\}$ , à la fois  $M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j)$  et  $M_{\rho_2}(i, j)$  sont égaux à 0.

Si  $j = k$  (respectivement  $i = k$ ), nous pouvons appliquer l'hypothèse d'induction à l'ensemble  $\mathcal{S}_{\rho_1}(i, 0)$  (respectivement  $\mathcal{S}_{\rho_1}(0, j)$ ). Comme plus haut, nous obtenons le résultat escompté.

Sinon, nous appliquons l'hypothèse d'induction à l'ensemble  $\mathcal{S}_{\rho_1}(i, j)$  et concluons de la même façon.

Finalement, nous obtenons dans chacun des cas que  $M_{\rho_2}(i, j) = M_{\rho_2[G_0 \leftarrow \text{tt}]}(i, j)$ , ce qui conclut la preuve.  $\square$

L'algorithme 4 retourne donc un ensemble d'ensembles de gardes diagonales lorsque le témoin s'avère être un faux témoin. Chacun de ces ensembles  $G$  contient des candidats pour le raffinement suivant du processus général. La proposition suivante établit que s'il existe un faux témoin dans un

automate temporisé, alors chaque ensemble  $G$  appartenant à l'ensemble retourné par l'algorithme 4 est non vide. Ceci prouve la correction de notre étape de raffinement : à chaque fois que nous devons raffiner le modèle, nous en sommes capables.

**Proposition 6.14.** *Soit  $\mathcal{A}$  un automate temporisé et  $\rho$  un faux témoin dans  $\mathcal{A}$ . Alors  $\text{Choisir\_gardes}(\rho)$  ne contient pas l'ensemble vide.*

**Preuve.** La preuve procède par l'absurde. Supposons que cet ensemble contienne l'ensemble vide. D'après le lemme 6.12, nous pouvons effacer l'ensemble des gardes diagonales apparaissant sur la trace  $\rho$  et obtenir la même zone  $Z_\rho^e = Z_{\rho[\text{Diag}(\mathcal{A}) \leftarrow \text{tt}]}^e$ . Ceci contredit alors la correction de l'algorithme 2 pour les automates sans gardes diagonales. En effet, notons  $\rho'$  la trace sans gardes diagonales définies par  $\rho' = \rho[\text{Diag}(\mathcal{A}) \leftarrow \text{tt}]$ . D'une part, nous avons  $\emptyset \neq Z_\rho^a \subseteq Z_{\rho'}^a$  puisque la trace  $\rho'$  est moins contraignante que la trace  $\rho$ . D'autre part, le lemme 6.12 donne  $\emptyset = Z_\rho^e = Z_{\rho'}^e$ , ce qui est absurde.  $\square$

La proposition suivante énonce la notion de progrès évoquée au début de cette section : pour tout faux-témoin  $\rho$  et tout ensemble  $G$  de gardes diagonales sélectionné par l'algorithme  $\text{Choisir\_gardes}$ ,  $\rho$  ne sera plus un témoin pour la nouvelle analyse de l'automate  $\mathcal{A}$  en tenant compte de la présence des éléments de  $G$ . Nous avons présenté deux méthodes permettant d'analyser l'automate vis-à-vis d'un certain ensemble de gardes diagonales. Pour l'une ou l'autre de ces deux méthodes, nous pouvons associer à une trace  $\rho$  dans l'automate temporisé initial un ensemble de traces noté  $\pi_{\mathcal{A},G}(\rho)$  correspondant à  $\rho$  dans la nouvelle analyse. En effet, si le raffiné de  $\mathcal{A}$  par rapport à  $G$  est construit explicitement, alors il est facile de constater que pour toute occurrence d'un élément de  $G$  dans  $\rho$ , deux traces  $\rho_{\text{tt}}$  et  $\rho_{\text{ff}}$  doivent être considérées selon que la garde est ou non satisfaite. De la même façon, la technique proposée dans [BY03] va produire deux successeurs, la zone intersectée avec  $\llbracket g \rrbracket$  et celle intersectée avec  $\llbracket \neg g \rrbracket$ , correspondant exactement aux deux traces précédentes. Notre résultat porte donc sur les traces éléments de  $\pi_{\mathcal{A},G}(\rho)$  :

**Proposition 6.15.** *Soit  $\mathcal{A}$  un automate temporisé et  $\rho$  un faux témoin dans  $\mathcal{A}$ . Considérons un élément  $G \in \text{Choisir\_gardes}(\rho)$ . Pour toute trace  $\rho' \in \pi_{\mathcal{A},G}(\rho)$ , le calcul des zones extrapolées prenant en compte  $G$  le long de  $\rho'$  mène à une zone vide, i.e.  $Z_{\rho'}^a = \emptyset$*

**Preuve.** La trace  $\rho$  est un faux témoin de  $\mathcal{A}$ . Elle s'écrit donc  $\rho = \rho_1.\alpha$  avec  $Z_{\rho_1}^e \neq \emptyset$ ,  $Z_\rho^e = \emptyset$  et  $Z_\rho^a \neq \emptyset$ . Supposons que le coefficient incorrect dans  $Z_{\rho_1}^a$  est le coefficient  $(l, k)$ . L'algorithme  $\text{Choisir\_gardes}$  retourne alors l'ensemble  $\mathcal{S}_{\rho_1}(l, k) \wedge \text{Ens}(g)$  où  $g$  est la garde étiquetant  $\alpha$ .

Considérons l'ensemble  $\overline{G} = \text{Diag}(\mathcal{A}) \setminus G$ . Le lemme 6.12 assure que la trace  $\rho_1[\overline{G} \leftarrow \text{tt}]$  mène à une zone ayant le même coefficient  $(l, k)$  via un calcul exact en avant :  $M_{\rho_1[\overline{G} \leftarrow \text{tt}]}(l, k) = M_{\rho_1}(l, k)$ . Le fait que la zone  $Z_\rho^e$  soit vide implique alors que la zone  $Z_{\rho_1[\overline{G} \leftarrow \text{tt}]}^e$  l'est également.

Soit  $\mathcal{A}_1$  l'automate temporisé obtenu à partir de  $\mathcal{A}$  en remplaçant toute garde de  $\overline{G}$  par  $\text{tt}$ . La trace  $\rho[\overline{G} \leftarrow \text{tt}]$  est une trace de  $\mathcal{A}_1$  et elle mène également à une zone vide, puisque le calcul exact en avant ne dépend que de la trace et non de l'automate dans lequel il est calculé.

Considérons à présent l'analyse de l'automate  $\mathcal{A}_1$  en raffinant par rapport aux gardes diagonales de  $G$ , selon l'une ou l'autre des deux méthodes présentées. Les seules gardes diagonales présentes dans  $\mathcal{A}_1$  sont celles de  $G$ , cette analyse revient donc à l'analyse d'un automate temporisé sans garde diagonale et est donc correcte pour l'accessibilité. Considérons alors le cas de la trace  $\rho[\overline{G} \leftarrow \text{tt}]$ . Les traces correspondant à l'analyse précédente sont les éléments de  $\pi_{\mathcal{A}_1,G}(\rho[\overline{G} \leftarrow \text{tt}])$ . Notons  $\rho_2$  un tel élément. Naturellement, puisque la zone  $Z_{\rho[\overline{G} \leftarrow \text{tt}]}^e$  est vide, la zone  $Z_{\rho_2}^e$  l'est également. L'analyse de  $\mathcal{A}_1$  en raffinant  $G$  correspondant à l'analyse d'un automate temporisé sans garde diagonale, la

correction de l'algorithme d'analyse en avant des automates temporisés sans garde diagonale implique donc  $Z_{\rho_2}^a = \emptyset$ .

Considérons enfin l'analyse de l'automate initial  $\mathcal{A}$  réalisée en raffinant par rapport à  $G$ , menant donc à l'ensemble de traces  $\pi_{\mathcal{A},G}(\rho)$ . Comme certaines gardes de  $\mathcal{A}$  ont été remplacées par tt dans l'automate  $\mathcal{A}_1$ , nous avons que toute trace  $\rho' \in \pi_{\mathcal{A},G}(\rho)$  est plus restrictive qu'une certaine trace  $\rho_2 \in \pi_{\mathcal{A}_1,G}(\rho[\overline{G} \leftarrow \text{tt}])$ . Nous obtenons donc, pour toute trace  $\rho' \in \pi_{\mathcal{A},G}(\rho)$ ,  $Z_{\rho'}^a = \emptyset$ . Ceci établit donc la propriété annoncée.  $\square$

Les deux propositions précédentes démontrent la correction de l'algorithme Choisir\_gardes pour la sélection des gardes diagonales lorsque le témoin est un faux témoin.

## 6.6 Implémentation dans UppAal

J'ai implémenté la méthode présentée dans les sections précédentes dans l'outil de vérification UPPAAL lors d'un séjour de deux semaines à Aalborg au Danemark en octobre 2005. Nous présentons ici la mise en œuvre de cette implémentation et les résultats obtenus.

### 6.6.1 Description de la structure du programme

**Structure de l'outil UPPAAL.** L'outil UPPAAL est constitué de deux modules principaux distincts. La première partie est une interface graphique utilisateur réalisée en Java. Celle-ci, très agréable pour l'utilisateur, lui permet de décrire aisément les systèmes qu'il souhaite étudier puis d'appliquer à ces objets différentes techniques de vérification. Ces appels utilisent la deuxième partie d'UPPAAL, à savoir son moteur de vérification sous-jacent, codé en C++. C'est cette partie qui contient la majeure difficulté, à savoir l'implémentation des algorithmes développés dans la communauté scientifique. Les structures de données mises en jeu sont complexes et de très nombreuses optimisations ont été apportées aux algorithmes existants qui font d'UPPAAL bien plus qu'une simple mise en œuvre des algorithmes standards de vérification pour les automates temporisés.

**Organisation de notre implémentation.** Depuis fin 2004, UPPAAL contient l'implémentation de la méthode proposée dans [BY03] décrite précédemment et qui réalise le raffinement du modèle par rapport à *toutes* les gardes diagonales du système. Cette implémentation nous sera utile à différents niveaux lors de nos travaux. D'autre part, UPPAAL permettait déjà de retourner la trace du contre-exemple obtenu lors de l'analyse de l'automate. Afin de développer notre technique, trois points étaient nécessaires :

- (i) un programme contrôlant la boucle générale de notre méthode,
- (ii) une méthode permettant de réaliser un raffinement partiel de l'automate vis-à-vis d'un sous-ensemble de contraintes diagonales,
- (iii) la méthode d'analyse des contre-exemples retournés par l'algorithme standard afin de sélectionner des gardes diagonales responsables des faux témoins.

Le premier point ne pose pas de difficultés. Pour le second point, nous avons choisi de nous appuyer sur l'implémentation existante et avons donc cherché à réutiliser la méthode de [BY03]. Pour cela, nous avons ajouté à la structure de l'algorithme un paramètre supplémentaire décrivant l'ensemble des gardes diagonales à prendre en compte lors du calcul de l'abstraction. Ceci nous permet de définir un opérateur d'extrapolation dépendant d'un ensemble de gardes diagonales et ainsi d'obtenir à peu de frais le point (ii). La majeure partie du travail a donc résidé dans le développement du point



(iii). Celui-ci a principalement nécessité le développement de la structure des matrices à différences bornées étendues avec les opérations associées. Dans un premier temps, ceci a consisté à définir une nouvelle classe représentant les seconds éléments des matrices à différences bornées étendues. Il a ensuite fallu développer les fonctions appropriées à ces nouveaux éléments. Enfin, pour s'adapter à la structure d'UPPAAL qui repose sur des notions d'héritage, nous avons défini une nouvelle classe représentant les états symboliques du système et prenant en compte une matrice à différences bornées étendue au lieu d'une matrice à différences bornées. En modifiant l'ensemble des opérateurs associés, cela nous a permis d'obtenir facilement l'algorithme de calcul en avant des successeurs le long de la trace comme une adaptation de l'algorithme existant à notre nouvelle classe.

### 6.6.2 Bancs de tests

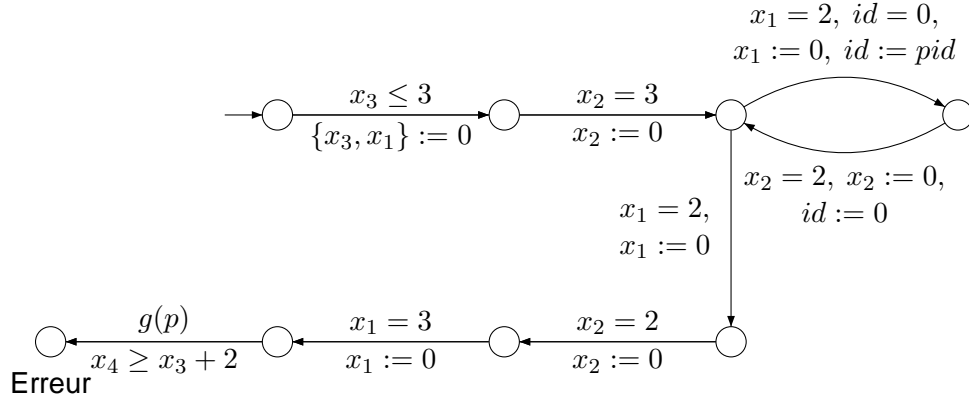
Nous présentons maintenant une série de tests réalisés afin d'évaluer les performances de notre algorithme. Pour obtenir une comparaison juste, nous avons considéré d'une part notre algorithme tel qu'il a été présenté précédemment et d'autre part l'algorithme existant réalisant la technique de [BY03]. Essentiellement trois cas peuvent survenir :

1. Aucun faux témoin n'existe et aucun raffinement du modèle n'est donc nécessaire.
2. Des faux témoins existent, le modèle doit donc être raffiné par rapport à certaines de ses gardes diagonales, mais pas par rapport à l'ensemble de celles-ci.
3. Des faux témoins existent et il apparaît nécessaire de raffiner le modèle par rapport à l'ensemble de ses gardes diagonales afin d'obtenir un algorithme correct.

Naturellement le premier cas est le plus favorable pour notre algorithme tandis que le dernier est le plus défavorable. Nous allons constater à l'aide des résultats suivants que les temps d'exécution de notre algorithme peuvent être nettement inférieurs à ceux de l'algorithme existant et que le surcoût de notre algorithme dans le cas défavorable reste négligeable.

Comme nous l'avons mentionné précédemment, les automates temporisés à l'origine de faux témoins sont très peu fréquents. En réalité, à notre connaissance, l'unique exemple existant est celui proposé dans [Bou04] représenté sur la figure 6.1. Afin d'obtenir des exemples provoquant de faux témoins, nous considérons donc un modèle dérivé de celui introduit dans [Bou04]. Nous considérons ensuite un exemple plus « réel », à savoir une modélisation du protocole de Fischer temporisé impliquant des gardes diagonales. UPPAAL ne permettant pas de donner le nombre de zones calculées lors de l'analyse en avant, nous donnons les mesures de temps (en secondes) et d'espace (en Mo) associées à l'exécution de l'algorithme. Nous donnons également, dans la colonne  $\#\mathcal{D}$ , le nombre de gardes diagonales prises en compte dans le raffinement du modèle. Ce nombre est toujours égal au nombre de gardes diagonales présentes dans le modèle pour la méthode [BY03].

**Un système provoquant des faux témoins.** Afin d'obtenir des systèmes de taille plus importante pour avoir des temps de calcul non nuls, nous construisons un réseau d'automates temporisés, chaque automate du réseau étant une « copie » de l'automate  $\mathcal{A}$  introduit dans [Bou04]. Nous introduisons donc un paramètre supplémentaire, le paramètre  $pid$ , dans l'automate temporisé de la figure 6.1. Ce paramètre permet d'identifier les automates du réseau. De plus, ces automates utilisent la variable partagée  $id$ . Celle-ci contrôle l'accès à la boucle, ce qui permet d'introduire des dépendances entre les différents automates. L'automate que nous considérons, noté  $\mathcal{A}_{id}^p$ , est représenté sur la figure 6.7.

FIG. 6.7 – Automate temporisé  $\mathcal{A}_{pid}^p$  dépendant du paramètre entier  $pid$ .

Celui-ci dépend également du paramètre  $p$  à travers la garde  $g(p)$ . Celle-ci est définie par :

$$g(p) = \begin{cases} x_2 \leq 1 & \text{si } p = 1 \\ x_2 - x_1 \leq 1 & \text{si } p = 2 \end{cases}$$

Ainsi, l'automate temporisé  $\mathcal{A}_{pid}^p$  contient  $p$  gardes diagonales. Nous allons dans la suite considérer les deux synchronisations suivantes :

$$\begin{aligned} \mathcal{A}_{1,2,3}^1 &= \mathcal{A}_1^1 \parallel \mathcal{A}_2^1 \parallel \mathcal{A}_3^1 \\ \mathcal{A}_{1,2,3}^2 &= \mathcal{A}_1^2 \parallel \mathcal{A}_2^2 \parallel \mathcal{A}_3^2 \end{aligned}$$

La variable partagée  $id$  est de type entier et appartient à l'intervalle  $[0, n]$  où  $n$  est le nombre de processus mis en jeu dans la composition. Ceci correspond au type de modèles pris en entrée par UPPAAL, mais il est facile de transformer ces modèles en des modèles équivalents dans le formalisme introduit dans le chapitre 1.

Nous nous intéressons alors à l'accessibilité de l'état **Erreur**. Plus précisément, nous considérons les problèmes d'accessibilité suivants, où  $L_i$  dénote les états de contrôle de l'automate  $\mathcal{A}_i^p$  :

$$\begin{aligned} Acc_i &= \{(\ell_1, \ell_2, \ell_3) \in L_1 \times L_2 \times L_3 \mid \ell_i = \text{Erreur}\} \\ Acc_{1,2} &= Acc_1 \cup Acc_2 \\ Acc_{1,2,3} &= Acc_1 \cup Acc_2 \cup Acc_3 \end{aligned}$$

Les résultats obtenus sont donnés dans la table 6.1. Comme cela était attendu, lorsque le nombre de raffinements réalisés par notre méthode est inférieur au nombre de gardes diagonales présentes dans le système, le gain en temps de calcul est très important. Ceci confirme donc l'intuition selon laquelle le raffinement vis-à-vis des diagonales peut être coûteux et qu'il vaut mieux l'éviter autant que possible. Cela contredit également l'affirmation de [BY03] selon laquelle le surcoût induit par le traitement des gardes diagonales est négligeable.

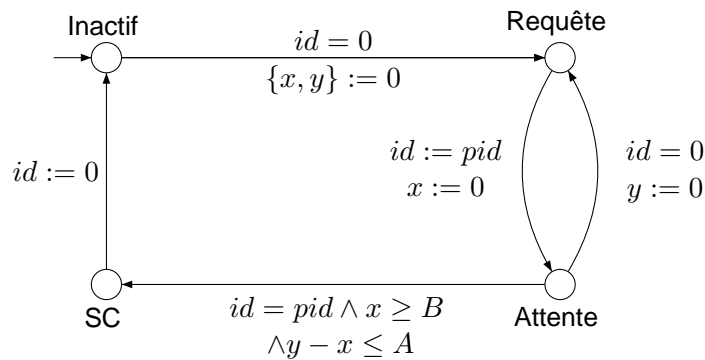
D'autre part, la dernière ligne de cette table illustre un cas intéressant, le cas (iii), le plus défavorable pour notre méthode. Ici encore, le résultat est satisfaisant puisque le surcoût de notre méthode, *i.e.* des différentes analyses partielles du système ainsi que de l'analyse des contre-exemples, est négligeable devant le coût total de l'algorithme (moins de 7% pour cet exemple).

$\mathcal{A}_{1,2,3}^2$	Notre algorithme			Algorithme [BY03]		
Requête	# $\mathcal{D}$	Temps	Espace	# $\mathcal{D}$	Temps	Espace
$Acc_1$	1	239,8	49,8	6	2080,4	70,0
$Acc_{1,2}$	2	465,2	53,8	6	2102,2	71,2
$Acc_{1,2,3}$	3	837,7	60,0	6	2110,0	70,7

$\mathcal{A}_{1,2,3}^1$	Notre algorithme			Algorithme [BY03]		
Requête	# $\mathcal{D}$	Temps	Espace	# $\mathcal{D}$	Temps	Espace
$Acc_1$	1	67,8	45,4	3	165,3	48,2
$Acc_{1,2}$	2	115,5	46,6	3	164,8	47,8
$Acc_{1,2,3}$	3	176,5	49,3	3	165,6	48,3

TAB. 6.1 – Résultats en présence de faux témoins.

FIG. 6.8 – Automate temporel  $\mathcal{P}_{id}$  dépendant du paramètre entier  $id$ .

**Protocole de Fischer.** Nous considérons une variante de la modélisation habituelle du protocole de Fischer temporel proposé dans [Zbr05] et qui introduit des contraintes diagonales. Le modèle représenté sur la figure 6.8, nommé  $\mathcal{P}_{pid}$ , dépend du paramètre entier  $pid$ , et de deux constantes  $A$  et  $B$ . Les systèmes considérés sont alors des compositions parallèles de différentes copies du même processus. À nouveau, les différents processus mis en jeu partagent la variable entière bornée  $id$ . Ainsi, lors de nos tests, nous considérons successivement la composition de 2, 3 puis 4 tels processus. Pour chaque processus, l'état de contrôle SC représente un accès à la section critique. Étant donné  $n$  processus mis en jeu, la question à laquelle nous nous intéressons est donc de savoir s'il peut arriver que 2 des  $n$  processus atteignent simultanément l'état SC. En notant  $L_i$  l'ensemble des états de contrôle du processus  $\mathcal{P}_{pid}$ , cette propriété s'énonce formellement comme l'accessibilité de l'ensemble suivant :

$$Acc = \{(\ell_i)_{1 \leq i \leq n} \in \prod_{1 \leq i \leq n} L_i \mid \exists 1 \leq i \neq j \leq n, \ell_i = \ell_j = SC\}$$

Cette propriété exprime que deux (ou plus) processus ont atteint simultanément leur section critique. Cet ensemble ne doit pas être accessible afin de garantir la propriété d'exclusion mutuelle. Il est alors possible de démontrer que cette propriété est vérifiée si et seulement si les constantes  $A$  et  $B$  vérifient  $A < B$ .

La table 6.2 donne les résultats des exécutions des deux algorithmes sur différentes instances du problème. Les valeurs de  $A$  et  $B$  valent respectivement 1 et 2. La première colonne donne le nombre de processus considérés dans la composition, les colonnes suivantes illustrent les résultats comme cela

a été fait dans la table précédente. À nouveau, le gain obtenu en appliquant notre algorithme est très important (un rapport 50 pour 4 processus).

Fischer	Notre algorithme			Algorithme [BY03]		
Nb de processus	$\#\mathcal{D}$	Temps	Espace	$\#\mathcal{D}$	Temps	Espace
2	0	0,01	1,4	4	0,01	1,4
3	0	0,02	1,4	6	0,42	38,3
4	0	11,6	40,4	8	560,8	50,2

TAB. 6.2 – Résultats pour le protocole de Fischer.

## 6.7 Conclusion

Nous nous sommes intéressés dans ce chapitre à l'analyse en avant à la volée dans les automates temporisés avec gardes diagonales. Après avoir décrit précisément dans la section 6.2 l'algorithme standard d'analyse en avant dans les automates temporisés (sans gardes diagonales), nous avons expliqué pourquoi celui-ci ne s'étendait pas directement au cadre des automates temporisés avec gardes diagonales dans la section 6.3. En particulier, nous avons montré qu'en présence de gardes diagonales, l'algorithme calcule une sur-approximation de l'ensemble des états accessibles qui s'avère être trop grossière pour décider le problème de l'accessibilité d'un état de contrôle. Nous avons ensuite étudié deux types de méthodes afin de proposer une correction de celui-ci, le retrait des gardes diagonales et la détection des faux-positifs. Cependant, raffiner l'automate vis-à-vis de l'ensemble des gardes diagonales présentes dans l'automate est trop coûteux du point de vue de la complexité et tester les contre-exemples afin de détecter les faux-positifs ne suffit pas à obtenir un algorithme correct et complet. Nous avons proposé une solution originale, fondée sur le concept du raffinement successif du modèle guidé par les contre-exemples retournés par l'analyse. Dans notre cadre, le raffinement du modèle correspond en fait à la prise en compte lors de l'analyse d'un certain sous-ensemble de gardes diagonales. Nous avons alors développé dans la section 6.5 un algorithme permettant, étant donné un faux-positif, de sélectionner un ensemble de gardes diagonales définissant le raffinement suivant du processus. Nous avons démontré la correction de notre méthode, ainsi qu'un critère de progrès vérifié par notre algorithme de sélection. Cette méthode a fait l'objet d'une implémentation dans l'outil UPPAAL présentée dans la section 6.6. Des tests ont également été présentés qui démontrent l'impact que peut avoir la présence de gardes diagonales sur la vérification des systèmes et les gains importants réalisés en utilisant notre algorithme. Ceci démontre l'intérêt de notre algorithme qui n'utilise l'opération de raffinement que lorsque cela est nécessaire et uniquement sur des gardes diagonales impliquées dans les erreurs commises par l'algorithme.

Plusieurs extensions de ces travaux sont envisageables. D'abord, il peut paraître intéressant de chercher à améliorer l'efficacité de notre algorithme. Pour cela, il semble prometteur par exemple de chercher à appliquer le concept de réduction d'horloges actives aux gardes diagonales. Ceci consiste à déterminer qu'une garde diagonale ne peut pas être responsable d'erreurs dans certains états de contrôle de l'automate. Il serait également intéressant d'étudier l'application à notre cadre d'une amélioration de la méthode de raffinements d'abstraction fondés sur l'étude des contre-exemples. En effet, dans [HJMS02], les auteurs proposent de réutiliser les calculs réalisés dans les analyses précédentes lors du raffinement de l'abstraction. Ceci peut avoir des conséquences importantes sur le comportement de l'algorithme. Une seconde voie consiste à appliquer les méthodes développées ici pour les

---

gardes diagonales à un cadre plus général. En effet, nous avons considéré un ensemble très restreint de prédicats (correspondant aux gardes diagonales) suffisant pour traiter notre problème. Il semble intéressant de chercher à appliquer d'une façon plus large les techniques d'abstraction fondée sur des prédicats [CGJ<sup>+</sup>00], ainsi que la notion de « raffinement paresseux » [HJMS02], comme une alternative à l'algorithme standard d'analyse en avant pour les automates temporisés.



## Chapitre 7

# Dépliage de réseaux d'automates temporisés

Lors de la présentation des automates temporisés dans le chapitre 1 et plus particulièrement des réseaux d'automates temporisés, nous avons souligné l'importance de développer des techniques pour l'analyse de systèmes temporisés tirant profit de leur caractère distribué. Nous avons également souligné le rôle joué par les invariants dans ces systèmes. Nous exposons dans ce chapitre le développement d'une technique de dépliages pour les réseaux d'automates temporisés avec partage d'horloges et invariants. La plupart des résultats présentés dans ce chapitre ont été publiés dans [BHR06d].

### 7.1 Introduction

**Motivations.** Les systèmes distribués constituent un cadre très favorable pour la modélisation car les systèmes réels présentent naturellement une structure distribuée. De plus, la modélisation d'un système comme la composition de différents processus permet d'obtenir des descriptions plus concises. Il est donc particulièrement intéressant de développer des techniques de vérification adaptées aux systèmes distribués. Au cours des dernières décennies, des avancées majeures ont été réalisées dans ce domaine. Les techniques développées peuvent être regroupées essentiellement dans deux catégories : les *méthodes d'ordres partiels* tirent principalement parti de l'indépendance entre les transitions (voir par exemple [Val89]), *i.e.* de transitions intervenant dans des composantes différentes du système. En particulier, les techniques de réductions d'ordres partiels utilisent cette indépendance afin de réduire le nombre d'entrelacements à considérer lors de l'exploration du graphe d'accessibilité du système. La seconde catégorie, à savoir les *méthodes de vraie concurrence*, couvre les méthodes fondées sur la construction d'un dépliage du système [NPW81, McM95]. Ces méthodes tirent à la fois profit de l'indépendance entre les transitions ainsi que de leur localité. De plus, les dépliages des systèmes constituent du point de vue théorique une sémantique concurrente alternative à la sémantique d'entrelacements habituelle. Soulignons que cette sémantique est plus discriminante que la sémantique classique et qu'elle peut être appliquée à d'autres domaines que la vérification comme l'observation ou le diagnostic [CJ05].

Dans le contexte des systèmes temporisés, la concurrence est une notion plus délicate et pose des difficultés dès la définition du modèle. Nous avons déjà étudié dans les chapitres 1, 2, 4 et 5 différents modèles combinant les aspects temporisés et distribués, à savoir différents types de réseaux d'automates temporisés et deux extensions des réseaux de Petri, les réseaux de Petri temporels et les réseaux de Petri temporisés. Cependant, nous avons toujours étudié ces modèles par rapport à

la sémantique des entrelacements perdant ainsi la notion de concurrence. Notre choix se porte ici sur le modèle des réseaux d'automates temporisés avec invariants et avec horloges partagées. Nous avons montré dans le chapitre 1 l'intérêt des invariants dans ce cadre (il n'est pas possible de s'en affranchir localement). De plus, les horloges partagées constituent une richesse supplémentaire lors de la modélisation. Ce modèle, couramment utilisé, offre une grande expressivité du point de vue des mécanismes temporels.

Du point de vue de l'analyse, le temps est également antagoniste à la notion de concurrence car il constitue un facteur de synchronisation entre les différents processus. En effet, tous les automates du réseau possèdent le même temps global et l'analyse doit donc s'assurer que cette propriété est satisfaite lors des transitions de délais (écoulement du temps). Combiné avec les invariants, ceci contraint fortement les automates du réseau entre eux. Ceci explique pourquoi ce domaine est difficile. En particulier, l'outil principal d'analyse pour les automates temporisés, *i.e.* l'outil UPPAAL, qui prend en entrée des réseaux d'automates temporisés proches de ceux que nous considérons, n'utilise aucune méthode d'ordres partiels lors de l'analyse du système. Au lieu de calculer l'automate produit équivalent, il se contente de l'explorer en le calculant à la volée. Il est donc intéressant de chercher à développer de nouveaux algorithmes tirant parti de la concurrence pour les réseaux d'automates temporisés.

**Travaux existants.** *Méthode d'ordres partiels pour les automates temporisés fondée sur les ensembles amples.* Lors de l'exploration des états du système, les méthodes d'ordres partiels fondées sur les ensembles amples sélectionnent un sous-ensemble de transitions au lieu de développer l'ensemble des transitions franchissables. Ce sous-ensemble, dénommé *ensemble ample*, vérifie certaines propriétés liées à la relation d'indépendance entre les transitions (voir [Pel93] pour plus de détails). L'efficacité de ces méthodes est donc étroitement liée à la taille de la relation d'indépendance. Ainsi, l'introduction du temps et des synchronisations implicites qui lui sont dues restreint nécessairement la relation correspondante pour le modèle non temporisé associé. Dans [BJLY98, Min99], les auteurs définissent une sémantique alternative pour les réseaux d'automates temporisés fondée sur un écoulement local du temps. Intuitivement, chaque automate peut laisser s'écouler le temps localement, sans tenir compte des autres automates. Ceci permet de réduire les dépendances introduites par le temps mais nécessite de resynchroniser les différentes dates locales lors des franchissements de transitions synchronisées. Bien que cette nouvelle sémantique autorise plus de comportements, la relation d'accessibilité associée à la sémantique classique peut être testée sur le système de transitions temporisé obtenu avec cette nouvelle sémantique. Ceci permet l'application de méthodes fondées sur les ensembles amples. En conclusion, l'efficacité de cette approche dépend de deux facteurs opposés : la sémantique de temps local engendre plus de comportements mais la relation d'indépendance, accrue pour cette nouvelle sémantique, permet de restreindre l'exploration de ces comportements.

*Méthode d'ordres partiels pour les automates temporisés fondée sur les traces de Mazurkiewicz.* Dans [LNZ04], deux transitions d'un automate temporisé sont indépendantes si elles commutent et si de plus chacune d'entre elles ne met pas à jour une horloge testée (ou mise à jour) par l'autre transition. L'indépendance est alors exploitée d'une façon différente : les occurrences de deux transitions indépendantes n'ont pas besoin d'être ordonnées (et par conséquent pas non plus les remises à zéro des horloges qui leur sont associées). Un état symbolique dans ce cadre est défini par un état de contrôle et une zone portant sur des variables correspondant aux remises à zéro et aux occurrences des transitions. Lorsque deux séquences  $ab$  et  $ba$  sont développées depuis un état symbolique avec  $a$  et  $b$  indépendantes, ces deux séquences mènent au même état symbolique tandis que la construction classique mènerait en général à deux états distincts. Cependant, cette méthode n'exploite pas la relation



d'indépendance afin de limiter l'exploration du système. Cette méthode a été étendue dans [NQ06a] de sorte à prendre en compte les invariants à l'aide d'une transformation syntaxique de l'automate produit associé au réseau d'automates temporisés. Les invariants ne sont donc pas traités au niveau local. Cette approche a été implémentée dans l'outil POEM (voir [NQ06b]) et présente des résultats encourageants.

*Méthodes d'ordres partiels pour les réseaux de Petri temporels fondées sur les ensembles amples ou « têtus ».* Dans les réseaux de Petri, les ensembles amples sont appelés « ensembles têtus » [Val89]. Ces ensembles sont similaires aux ensembles amples mais leur définition tire profit de la localité de la condition de franchissement d'une transition. Dans [YS97], les auteurs généralisent ce concept aux réseaux de Petri temporels, nomment ces ensembles des « ensembles parés », et l'appliquent à la construction du graphe des classes de [BD91] (voir page 61 pour une présentation de cette construction). Étant donné un état symbolique, un ensemble paré est constitué d'un ensemble têtus et d'une contrainte supplémentaire portant sur les conditions temporelles des transitions actives. L'efficacité de la méthode repose fortement sur la faiblesse du couplage temporel existant entre les transitions.

*Méthode d'ordres partiels pour les réseaux de Petri temporels fondée sur les dépliages.* Selon le réseau de Petri analysé, les méthodes fondées sur les dépliages et sur les ensembles têtus se comportent très différemment. Ainsi, la première est plus performante que la seconde pour les réseaux présentant de la confusion, *i.e.* lorsque le tir d'une transition peut influencer les places permettant le franchissement d'une autre transition du réseau. La généralisation de la construction d'un dépliage aux réseaux de Petri temporels a suscité de nombreux travaux. D'abord, [Li98] a proposé une méthode permettant de contrôler la construction du graphe des classes en utilisant un dépliage du réseau de Petri non temporisé sous-jacent. Cependant, ce dépliage peut être infini même si le réseau de Petri temporel est borné<sup>1</sup>. Puis, dans [AL00] les auteurs ont montré que la sémantique urgente des réseaux de Petri temporels nécessite une analyse globale du processus afin de tester le franchissement d'une transition dans ce processus. Plus précisément, l'urgence présente dans ce modèle globalise les dépendances temporelles, ce qui constitue un résultat négatif pour la construction du dépliage, puisque celle-ci doit être faite localement. Suivant ces remarques, [CJ06] a récemment développé une méthode permettant de construire un dépliage du système en introduisant des arcs de lecture afin d'exprimer les dépendances temporelles supplémentaires. Cette méthode permet de définir un tel objet, et d'en exhiber un préfixe fini et complet mais le coût de la construction reste prohibitif, comme annoncé dans [AL00]. Enfin, dans une direction différente, [FS02] propose une sémantique à temps discret équivalente à la sémantique à temps dense du point de vue de l'accessibilité. Le réseau inclut une transition spéciale représentant l'écoulement du temps mais toute occurrence de cette transition requiert une synchronisation globale ce qui réduit de façon très importante la localité du dépliage. De plus, cette approche souffre de l'explosion combinatoire apparaissant nécessairement dans toute approche à temps discret.

**Contributions.** Nous nous intéressons dans ce chapitre au développement d'un algorithme efficace de vérification pour les réseaux d'automates temporisés avec invariants et horloges partagées. Notre algorithme construit un dépliage du réseau analysé sous la forme d'un réseau d'occurrences contextuel, *i.e.* de façon approximative un réseau de Petri acyclique étendu avec des arcs de lecture. Les conditions de ce réseau d'occurrences (les places du réseau de Petri) sont étiquetées par des états de contrôle ou des horloges du réseau d'automates temporisés. Les événements du réseau d'occurrences (les transitions du réseau de Petri) sont étiquetés par les transitions globales du réseau d'automates temporisés. Les arcs de lecture permettent de modéliser les tests réalisés sur les horloges (voir le chapitre 5) et d'exprimer les dépendances vis-à-vis des invariants. Bien que ces arcs n'augmentent pas le

<sup>1</sup>Rappelons que le réseau de Petri sous-jacent à un réseau de Petri temporel peut être non borné même si celui-ci l'est.

pouvoir d'expression des réseaux de Petri (un arc de lecture peut être simulé par une consommation et une production) vis-à-vis de la sémantique d'entrelacements, ils accroissent le pouvoir d'expression vis-à-vis d'une sémantique concurrente [MR95]. En particulier, ils permettent d'accroître la relation d'indépendance entre les événements.

Plus précisément, nous définissons un dépliage temporisé d'un réseau d'automates temporisés relativement proche du dépliage classique du cadre non temporisé [McM95, ERV02]. La différence majeure consiste à associer à chaque événement du dépliage en plus du marquage lui correspondant une zone. Intuitivement, la zone attachée à un événement  $t$  décrit l'ensemble des valeurs d'horloges accessibles à l'issue du franchissement de  $t$  et de tous les événements nécessaires au tir de  $t$ . Notre dépliage est donc symbolique au sens où chaque processus non branchant représente, via ces zones, un nombre infini de séquences temporisées du réseau d'automates temporisés.

Le problème majeur rencontré par les travaux précédents réside dans l'urgence, due dans les automates temporisés aux invariants, qui induit des dépendances temporelles globales entre des transitions *a priori* indépendantes, comme cela a été mis en évidence dans [AL00] dans le cadre des réseaux de Petri temporels. Lorsqu'une horloge apparaît dans un invariant, nous utilisons des arcs de lecture pour exprimer ces dépendances. Ceci nous permet d'accroître la relation de concurrence entre les événements tout en conservant un processus de décision local du franchissement d'un événement : il est suffisant de prendre en compte les conditions « en entrée » de l'événement pour décider si celui-ci peut ou non être franchi. Nous maintenons ainsi la caractéristique principale des dépliages en tirant profit à la fois de la concurrence entre les événements et de leur localité.

Nous proposons également une optimisation de notre algorithme n'utilisant lors du calcul que des zones de dimension constante égale à trois fois le nombre d'horloges du réseau plus le nombre d'automates dans le réseau. Cette optimisation peut se révéler primordiale car les opérations standards sur les zones comme le test du vide sont cubiques dans le nombre de variables.

Enfin, nous développons deux approches sensiblement différentes permettant de construire un préfixe fini et complet du dépliage temporisé défini précédemment. La première repose sur les travaux de [Win02] qui permettent d'exhiber un préfixe fini et complet pour les dépliages des réseaux de Petri saufs avec arcs de lecture tandis que la seconde peut introduire des synchronisations supplémentaires mais permet d'obtenir un algorithme plus simple. Dans les deux cas, ces préfixes finis permettent de décider en temps linéaire (dans la taille du préfixe) l'accessibilité d'une configuration ainsi que la possibilité de tirer une transition.

**Organisation du chapitre.** Dans la section 7.2, afin de familiariser le lecteur avec les notions mises en jeu dans les dépliages, nous présentons la construction du dépliage d'un réseau d'automates finis. Nous illustrons ensuite avec différents exemples qui seront réutilisés tout au long du chapitre les principales difficultés rencontrées lors de la prise en compte des contraintes temporelles, motivant ainsi le choix de représenter explicitement les horloges à l'aide de conditions et d'introduire des arcs de lecture. Nous présentons ensuite dans la section 7.3 le dépliage non temporisé (*i.e.* sans les zones) d'un réseau d'automates temporisés. La section 7.4 est dédiée à l'extension de ce dépliage à l'aide de zones, conduisant à la définition du dépliage temporisé d'un réseau d'automates temporisés. Nous y présentons également l'optimisation permettant un calcul de ce dépliage n'utilisant que des zones de dimension constante. Nous présentons dans la section 7.5 deux algorithmes pour la construction d'un préfixe fini et complet du dépliage temporisé d'un réseau d'automates temporisés. Enfin, la section 7.6 présente une discussion sur notre méthode esquissant de possibles simplifications et/ou adaptations à des sous-classe du modèle que nous avons considéré et la section 7.7 conclut ce chapitre.

## 7.2 Préliminaires

Nous présentons dans cette section le cadre des dépliages de réseaux d'automates finis et l'appliquons à deux exemples afin d'illustrer les difficultés lors de l'extension au cadre des réseaux d'automates temporisés.

### 7.2.1 Dépliages de réseaux d'automates finis

Nous présentons les notions élémentaires relatives à la définition et à la construction de dépliages pour des réseaux d'automates finis puisque le cadre que nous étudions, les réseaux d'automates temporisés, en constitue une extension. Nous ne donnons pas la définition formelle d'un réseau d'automates finis qui peut aisément être déduite de la définition donnée pour les réseaux d'automates temporisés.

Le dépliage d'un réseau d'automates finis consiste en un réseau de Petri d'un type particulier, appelé réseau d'occurrences [McM95, ERV02]. Avant d'introduire cette nouvelle classe, nous devons définir certaines notions propres au réseau de Petri étudié. Nous considérons donc un réseau de Petri  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$ .

**Causalité** Nous définissons la *relation causale*  $<$  sur l'ensemble des nœuds  $P \cup T$  de  $\mathcal{N}$  comme la relation transitive minimale sur  $P \cup T$  telle que pour tout  $t \in T, p \in P$  :

- $p \in \bullet t \Rightarrow p < t$ ,
- $p \in t^\bullet \Rightarrow t < p$ .

Nous notons  $\leq$  la fermeture réflexive de  $<$ .

**Conflit** Nous définissons la *relation de conflit*  $\#$  sur l'ensemble des nœuds de  $\mathcal{N}$  par  $x \# x'$  si et seulement si il existe une place  $p \in P$  et deux transitions  $t, t' \in T$  telles que :

- $p \in \bullet t \cap \bullet t'$ ,
- $t \neq t'$ ,
- $t \leq x$  et  $t' \leq x'$ .

**Concurrence** Nous définissons la *relation de concurrence*  $co$  sur l'ensemble des nœuds de  $\mathcal{N}$  par  $x co y$  si et seulement si ni  $x < y$ , ni  $y < x$ , ni  $x \# y$  n'est vérifié.

**Places minimales** Nous définissons enfin l'ensemble des places minimales de  $\mathcal{N}$ , noté  $Min(\mathcal{N})$ , par  $Min(\mathcal{N}) = \{p \in P \mid \bullet p = \emptyset\}$ .

Nous définissons à présent la notion de réseau d'occurrences obtenue comme une sous-classe des réseaux de Petri. Dans un réseau d'occurrences, les places seront appelées « conditions » et les transitions « événements ».

**Définition 7.1** (Réseau d'occurrences). *Un réseau d'occurrences est un réseau de Petri<sup>2</sup>  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  vérifiant les conditions suivantes :*

- (i)  $|\bullet p| \leq 1$  pour toute condition  $p \in P$ ,
- (ii)  $\mathcal{N}$  est acyclique, i.e. la relation causale  $<$  est un ordre partiel,
- (iii)  $<$  est finiment précédée, i.e. pour tout nœud  $x \in P \cup T$ , l'ensemble  $\{y \in P \cup T \mid y < x\}$  est fini,
- (iv) aucun nœud n'est en conflit avec lui-même : pour tout  $x \in P \cup T$ , nous n'avons pas  $x \# x$  et
- (v) le marquage  $M_0$  vérifie  $M_0 = Min(\mathcal{N})$ .

<sup>2</sup>Nous autorisons de plus les ensembles de places et de transitions à être infinis.

Nous introduisons également quelques notations nécessaires à la description d'un produit synchronisé d'automates finis. Étant donné un réseau d'automates finis  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$ , où chaque  $\mathcal{A}_i$  est un automate fini décrit par  $\mathcal{A}_i = (\Sigma, L_i, T_i, \ell_{i,0})$ , nous notons  $L = \bigcup_{1 \leq i \leq n} L_i$  l'ensemble des états de contrôle des automates. L'ensemble des transitions synchronisées de  $\mathcal{A}$ , sans tenir compte de la fonction de synchronisation  $f$  est noté  $\overline{T}$  et défini par

$$\overline{T} = \prod_{1 \leq i \leq n} (T_i \cup \{\perp\}) \setminus \{\perp^n\}$$

Seuls les éléments de  $\overline{T}$  qui correspondent à une synchronisation autorisée par  $f$  peuvent réellement survenir. Étant donné un élément  $\bar{t} = (t_i)_{1 \leq i \leq n} \in \overline{T}$ , nous définissons le sous-ensemble  $I(\bar{t})$  de  $\{1, \dots, n\}$  des indices des automates mis en jeu par  $\bar{t}$  par  $I(\bar{t}) = \{i \mid t_i \neq \perp\}$  et nous écrivons  $t_i = (\ell_i, a_i, \ell'_i)$  pour tout  $i \in I(\bar{t})$ . Nous posons  $a_i = \perp$  si  $i \notin I(\bar{t})$ . Nous définissons alors l'ensemble des synchronisations possibles, noté  $Sync$ , par :

$$Sync = \{\bar{t} \in \overline{T} \mid \exists a \in \Sigma \text{ tel que } f(a_1, \dots, a_n) = a\}$$

Étant donné un élément  $\bar{t} = (t_i)_{1 \leq i \leq n} \in Sync$ , nous définissons alors :

$$\begin{cases} Eti(\bar{t}) &= f((a_i)_{1 \leq i \leq n}) \\ Cons(\bar{t}) &= \bigcup_{i \in I(\bar{t})} \ell_i \\ Prod(\bar{t}) &= \bigcup_{i \in I(\bar{t})} \ell'_i \end{cases}$$

Nous définissons à présent un processus branchant associé à un réseau d'automates finis.

**Définition 7.2** (Processus branchant d'un réseau d'automates finis). *Soit  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$  un réseau d'automates finis. Un processus branchant de  $\mathcal{A}$  est une paire  $\beta = (\mathcal{N}, \mu)$  constituée d'un réseau d'occurrences  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)\bullet, M_0, \Lambda)$  et d'une application  $\mu$  de  $P \cup T$  dans  $L \cup Sync$  vérifiant les conditions suivantes :*

- $\mu(P) \subseteq L$ ,
- $\mu(T) \subseteq Sync$ ,
- $\mu$  est une bijection entre  $M_0$  et  $\{\ell_{i,0} \mid 1 \leq i \leq n\}$ ,
- pour tout  $t \in T$ ,  $\mu$  est une bijection entre  $\bullet t$  et  $Cons(\mu(t))$  et entre  $t\bullet$  et  $Prod(\mu(t))$ ,
- il n'y a pas de redondance, i.e. pour deux événements  $t, t' \in T$ , si  $\mu(t) = \mu(t')$  et  $\bullet t = \bullet t'$ , alors  $t = t'$ ,
- l'étiquetage de  $T$  est cohérent : pour tout  $t \in T$ , nous avons  $\Lambda(t) = Eti(\mu(t))$ .

Intuitivement, les processus branchants d'un même réseau d'automates finis diffèrent selon combien ils déplient le réseau, i.e. selon la longueur des exécutions qu'ils représentent. En particulier, ceci permet de définir un ordre partiel sur l'ensemble des processus branchants d'un réseau d'automates finis. Il est alors possible de démontrer que tout réseau d'automates finis  $\mathcal{A}$  possède un unique processus branchant maximal pour cet ordre, à isomorphisme près. C'est cet élément qui est appelé le dépliage de  $\mathcal{A}$ .

Naturellement, le dépliage d'un réseau d'automates finis peut être infini et il est donc nécessaire par la suite de s'intéresser à la définition d'un préfixe fini de celui-ci qui soit complet, i.e. qui représente toutes les configurations accessibles. L'existence d'un tel préfixe a été démontrée dans [McM95]. Cependant, ce préfixe n'est pas unique et il est préférable de choisir le plus petit, au sens du nombre de nœuds, afin d'obtenir la représentation la plus concise du réseau. Ces questions ont été étudiées dans [ERV02] et nous les aborderons dans la partie 7.5.

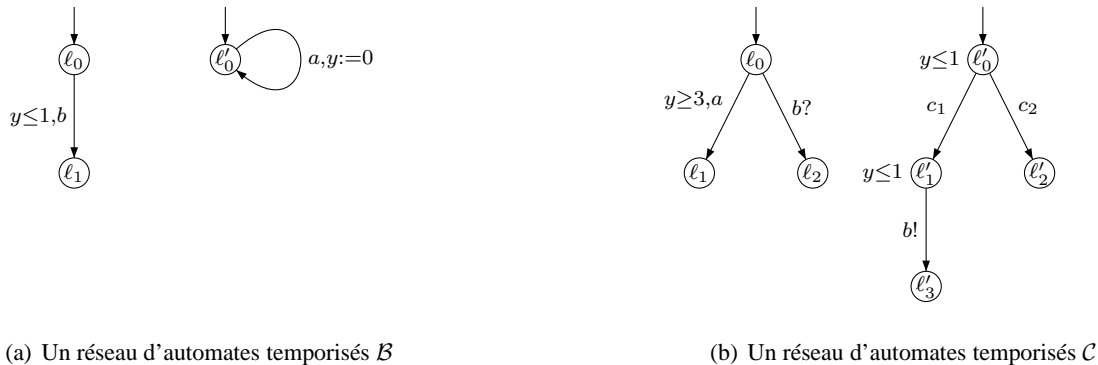
(a) Un réseau d'automates temporisés  $\mathcal{B}$ (b) Un réseau d'automates temporisés  $\mathcal{C}$ 

FIG. 7.1 – Deux réseaux d'automates temporisés.

### 7.2.2 Exemples explicatifs

Nous présentons dans cette sous-section deux exemples de réseaux d'automates temporisés. Nous donnons ensuite les réseaux d'automates finis qui leur sont naturellement associés et les dépliages correspondants. Ces exemples illustrent les définitions précédentes et mettent en lumière les limites des dépliages précédents pour l'analyse temporelle des réseaux. Ces deux exemples sont représentés sur la figure 7.1.

**Exemple 7.3** (Étude du réseau  $\mathcal{B}$  représenté sur la sous-figure 7.1(a)). Dans ce réseau, les deux transitions  $a$  et  $b$  ne sont pas synchronisées et les deux automates partagent l'horloge  $y$ . Ce réseau accepte les transitions  $b$  ayant lieu au plus une unité de temps après  $a$  (ou après le début de l'exécution). En effet, initialement l'horloge  $y$  est nulle et elle est remise à zéro à chaque franchissement de  $a$ .

À titre d'illustration, le réseau d'automates finis  $\mathcal{B}'$  associé à  $\mathcal{B}$  en retirant les gardes, invariants et remises à zéro et un préfixe de son dépliage sont représentés sur la figure 7.2. Sur ce préfixe, une seule occurrence de  $b$  est représentée et dans tout autre préfixe du dépliage de  $\mathcal{B}'$ , aucune autre occurrence de  $b$  n'existe. En particulier, les occurrences supplémentaires de  $a$  ne donnent pas lieu à de nouvelles occurrences de  $b$ . Ceci ne reflète pas la dépendance temporelle entre  $a$  et  $b$  énoncée précédemment. Afin de représenter dans le dépliage cette dépendance, il est donc naturel d'enrichir ce dépliage afin que les nouvelles occurrences de  $a$  soient à l'origine de nouvelles occurrences de  $b$ .  $\lrcorner$

**Exemple 7.4** (Étude du réseau  $\mathcal{C}$  représenté sur la sous-figure 7.1(b)). Dans ce réseau, les transitions  $a$ ,  $c_1$  et  $c_2$  ne sont pas synchronisées tandis que les transitions  $b?$  et  $b!$  le sont. De plus, la transition  $a$  ne peut survenir que si la transition  $c_2$  a eu lieu. En effet, les états de contrôle  $l'_0$  et  $l'_1$  sont contraints par l'invariant  $y \leq 1$  qui rend impossible le tir de  $a$  à cause de la garde  $y \geq 3$  associée au franchissement de  $a$ . Au contraire l'état de contrôle  $l'_2$  ne porte pas d'invariant. Le tir de  $a$  n'est donc possible que si le deuxième automate se trouve dans l'état  $l'_2$ , ce qui ne peut être le cas qu'après le tir de  $c_2$ .

Comme pour l'exemple précédent, le réseau d'automates finis  $\mathcal{C}'$  associé à  $\mathcal{C}$  en retirant les gardes, invariants et remises à zéro et son dépliage sont représentés sur la figure 7.2. Sur ce dépliage, rien n'indique la propriété de causalité énoncée précédemment entre  $a$  et  $c_2$ . Dans ce cas, les contraintes temporelles du réseau induisent donc une nouvelle causalité supplémentaire absente du dépliage du réseau non temporisé associé. Il est donc nécessaire d'enrichir ce dépliage afin de détecter ces contraintes temporelles. Nous allons augmenter l'ensemble des places en entrée de la transition  $t_1$  afin de tenir compte des dépendances dues aux invariants.  $\lrcorner$

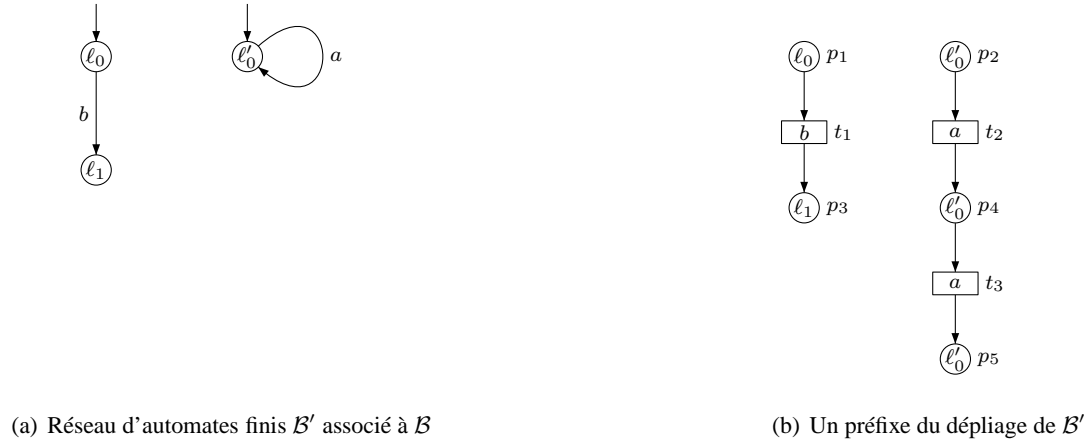


FIG. 7.2 – Dépliage du réseau d'automates finis associé à  $\mathcal{B}$ .

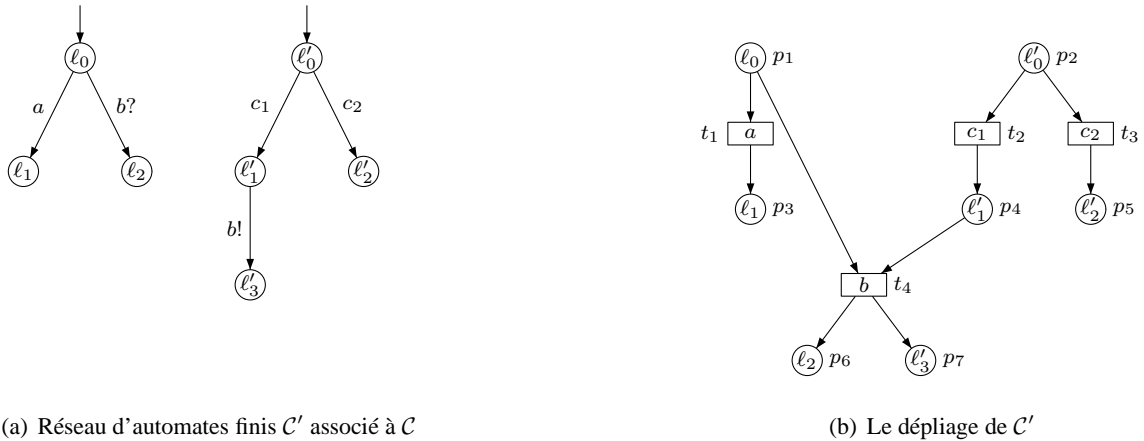


FIG. 7.3 – Dépliage du réseau d'automates finis associé à  $\mathcal{C}$ .

À la lumière des deux exemples précédents, la structure du dépliage du réseau d'automates finis sous-jacent au réseau d'automates temporisés étudié apparaît insuffisante pour représenter les dépendances supplémentaires introduites par les horloges et les invariants. Nous choisissons donc d'étendre le dépliage à l'aide des deux caractéristiques fondamentales suivantes :

- Nous ajoutons des places afin de représenter explicitement les horloges,
- Nous ajoutons des arcs de lecture au dépliage afin d'explicitier les dépendances temporelles.

La nature même de l'objet que nous allons construire sera donc différente, puisque le dépliage intégrera des arcs de lecture. Nous décrivons comment définir un tel dépliage dans la section suivante.

### 7.3 Dépliage non temporisé

Nous définissons dans cette section la structure discrète de notre dépliage. Nous présenterons dans la section suivante comment intégrer les contraintes temporisées à cette structure.

### 7.3.1 Réseau d'occurrences contextuel

Nous construisons un dépliage intégrant des arcs de lecture, ce qui nécessite de nouvelles définitions pour les relations causales, de conflit et de concurrence. Ces questions ont été étudiées dans [VSY98, Win02] et leurs travaux ont mis en évidence les difficultés liées à l'introduction d'arcs de lecture dans les dépliages. Cependant, les arcs de lecture possèdent de nombreux avantages que nous résumons ainsi :

- dans la définition de systèmes, ils permettent d'obtenir des sémantiques concurrentes plus raffinées [MR95],
- pour la construction d'un dépliage, ils augmentent l'ensemble des conditions dont dépend un événement,
- pour la construction d'un dépliage, ils expriment l'accès concurrent de deux événements à une même condition.

**Définition 7.5** (Réseau de Petri avec arcs de lecture). *Un réseau de Petri avec arcs de lecture  $\mathcal{N}$  est un uplet  $(P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  où :*

- $(P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, M_0, \Lambda)$  est un réseau de Petri, selon la définition 2.1, et
- $\circ(\cdot) \in \text{MEns}(P)^T$  est l'application d'incidence de lecture (correspondant aux arcs de lecture).

Le comportement d'un réseau de Petri avec arcs de lecture est l'extension naturelle du comportement d'un réseau de Petri. Les arcs de lecture jouent un rôle similaire aux arcs de lecture introduits dans les réseaux de Petri temporisés. Nous ne donnons donc pas la sémantique formelle des réseaux de Petri avec arcs de lecture, d'autant qu'elle ne nous sera pas utile.

Nous devons en revanche adapter les définitions de causalité, conflit et concurrence définies pour les réseaux de Petri. En effet, comme nous l'avons rappelé plus haut, les actions de lecture induisent de la richesse du point de vue de la sémantique concurrente du réseau. Afin de prendre en compte la présence des arcs de lecture, la relation causale est remplacée par deux relations, la relation de causalité faible et la relation de causalité forte. La relation de conflit  $\#$  et les places minimales  $Min$  sont identiques à celles définies pour les réseaux de Petri. La relation de concurrence sera présentée ultérieurement.

**Causalité faible** Nous définissons la *relation causale faible*  $<$  sur l'ensemble des nœuds  $P \cup T$  de  $\mathcal{N}$  comme la relation transitive minimale sur  $P \cup T$  telle que pour tout  $t, t' \in T, p \in P$  :

- $p \in \bullet t \Rightarrow p < t$ ,
- $p \in t^\bullet \Rightarrow t < p$ ,
- $p \in t^\bullet \wedge p \in \circ t' \Rightarrow t < t'$ .

Nous notons  $\leq$  la fermeture réflexive de  $<$ .

**Causalité forte** Nous définissons la *relation causale forte*  $\prec$  sur l'ensemble des nœuds  $P \cup T$  de  $\mathcal{N}$  comme la relation transitive minimale sur  $P \cup T$  telle que pour tout  $t, t' \in T, p \in P$  et pour tous nœuds  $x$  et  $y$  :

- $x < y \Rightarrow x \prec y$ ,
- $p \in \circ t \wedge p \in \bullet t' \Rightarrow t \prec t'$ .

Nous notons  $\preceq$  la fermeture réflexive de  $\prec$ .

**Conflit** Nous définissons la *relation de conflit*  $\#$  sur l'ensemble des nœuds de  $\mathcal{N}$  par  $x \# x'$  si et seulement s'il existe une place  $p \in P$  et deux transitions  $t, t' \in T$  telles que :

- $p \in \bullet t \cap \bullet t'$ ,
- $t \neq t'$ ,
- $t \leq x$  et  $t' \leq x'$ .



FIG. 7.4 – Réseaux d'occurrences contextuels, exemple et contre-exemple.

**Places minimales** Nous définissons enfin l'ensemble des places minimales de  $\mathcal{N}$ , noté  $Min(\mathcal{N})$ , par  $Min(\mathcal{N}) = \{p \in P \mid \bullet p = \emptyset\}$ .

Commentons les nouvelles notions de causalité faible et forte. Intuitivement, étant donné deux événements  $t$  et  $t'$ , nous avons  $t < t'$  si et seulement si le franchissement de l'événement  $t$  est nécessaire à celui de  $t'$ . En particulier, ceci implique que  $t$  apparaît dans toute histoire de  $t'$ . Pour la relation de causalité forte, nous avons  $t \prec t'$  si et seulement si pour tout franchissement de  $t$  et  $t'$ , le franchissement de  $t$  doit précéder celui de  $t'$ . En particulier, il existe des histoires de  $t'$  qui ne contiennent pas l'événement  $t$ , mais pour toute séquence de transitions contenant à la fois  $t$  et  $t'$ , alors  $t$  précède  $t'$ . Ceci implique que plusieurs « histoires » menant à l'événement  $t'$  peuvent exister. Enfin, remarquons que la causalité faible implique la causalité forte, *i.e.* que la relation de causalité faible est incluse dans la relation de causalité forte.

Un nouveau type de conflits apparaît avec la présence d'arcs de lecture. Il s'agit des *conflits contextuels*, dus aux cycles contextuels. Un *cycle contextuel* est un ensemble  $\{t_1, \dots, t_k\}$  d'événements tels que  $t_i \prec t_{i+1}$  pour tout  $i$  et de plus  $t_k \prec t_1$ . Cette situation est expliquée dans l'exemple 7.7. Ces cycles correspondent à des ensembles de transitions qui ne peuvent pas appartenir à une même exécution du système. Nous devons donc les interdire dans la définition des réseaux d'occurrences contextuels.

**Définition 7.6** (Réseau d'occurrences contextuel). *Un réseau d'occurrences contextuel est un réseau de Petri avec arcs de lecture<sup>3</sup>  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  vérifiant les conditions suivantes :*

- (i)  $|\bullet p| \leq 1$  pour toute condition  $p \in P$ ,
- (ii)  $\mathcal{N}$  est acyclique, *i.e.* la relation causale  $<$  est un ordre partiel,
- (iii)  $<$  est finiment précédée, *i.e.* pour tout nœud  $x \in P \cup T$ , l'ensemble  $\{y \in P \cup T \mid y < x\}$  est fini,
- (iv) pour tout nœud  $x$ , la restriction de  $<$  à l'ensemble des prédécesseurs de  $x$  pour  $<$  est un ordre partiel,
- (v) aucun nœud n'est en conflit avec lui-même : pour tout  $x \in P \cup T$ , nous n'avons pas  $x \# x$  et
- (vi) le marquage  $M_0$  vérifie  $M_0 = Min(\mathcal{N})$ .

<sup>3</sup>Nous autorisons de plus les ensembles de places et de transitions à être infinis.



Les points (i), (ii), (iii), (v) et (vi) sont similaires aux conditions requises dans la définition d'un réseau d'occurrences. La seule différence réside dans le point (iv). Celui-ci impose que pour tout nœud  $x$ , aucun conflit contextuel n'existe dans le passé causal faible de  $x$ , assurant ainsi le franchissement de  $x$  (si  $x$  est un événement).

**Exemple 7.7** (Réseaux d'occurrences contextuels). Considérons les deux réseaux représentés sur la figure 7.4. Le réseau représenté sur la sous-figure 7.4(a) constitue bien un réseau d'occurrences contextuel. Notons que nous avons  $t_1 < t_3$  et  $t_3 < t_2$  ce qui implique par transitivité  $t_1 < t_2$ . Considérons maintenant le réseau représenté sur la sous-figure 7.4(b). Ce réseau ne constitue pas un réseau d'occurrences contextuel. En effet, pour la relation  $<$ , les deux transitions  $t_1$  et  $t_2$  ne sont pas ordonnées mais, pour la relation  $<$ , nous avons à la fois  $t_1 < t_2$  et  $t_2 < t_1$ . Ceci rend impossible le franchissement de  $t_1$  et  $t_2$  dans une même exécution du réseau. Le point (iv) de la définition précédente n'est donc pas vérifié pour l'événement  $t_3$ . En revanche, si nous considérons le sous-réseau constitué des nœuds  $\{p_1, p_2, p_3, p_4, t_1, t_2\}$ , alors celui-ci est un réseau d'occurrences contextuel.  $\square$

### 7.3.2 Processus branchant d'un automate temporisé

Nous définissons à présent à la notion de processus branchant associé à un automate temporisé. Pour cela, nous réutilisons certaines définitions introduites plus haut pour les réseaux d'automates finis.

Étant donné un réseau d'automates temporisés  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$  où chaque  $\mathcal{A}_i$  est un automate temporisé décrit par<sup>4</sup>  $\mathcal{A}_i = (\Sigma, X_i, L_i, T_i, Inv_i, \ell_{i,0})$ , nous introduisons les notations suivantes<sup>5</sup> :

$$\begin{cases} L & = \cup_{1 \leq i \leq n} L_i \\ X & = \cup_{1 \leq i \leq n} X_i \\ X_{inv} & = \{x \in X \mid \exists i, \ell_i \in L_i, x \in \text{Horloges}(Inv_i(\ell_i))\} \end{cases}$$

Nous utilisons à nouveau les notations  $\bar{T}, \bar{t}$ , etc. introduites pour les réseaux d'automates.  $I(\bar{t})$  représente l'ensemble des indices des automates mis en jeu par la synchronisation  $\bar{t}$  et pour tout  $i \in I(\bar{t})$ , nous écrivons  $t_i = (\ell_i, g_i, a_i, R_i, \ell'_i)$ . Enfin, nous posons  $a_i = \perp$  si  $i \notin I(\bar{t})$ . L'ensemble des transitions synchronisées de  $\mathcal{A}$ , noté  $\text{Sync}$ , est alors défini par

$$\text{Sync} = \{\bar{t} \in \bar{T} \mid \exists a \in \Sigma \text{ tel que } f(a_1, \dots, a_n) = a\}$$

Nous définissons alors les ensembles suivants<sup>6</sup> :

$$\begin{cases} \text{Eti}(\bar{t}) & = f((a_i)_{1 \leq i \leq n}) \\ g(\bar{t}) & = \bigwedge_{i \in I(\bar{t})} g_i \\ R(\bar{t}) & = \bigcup_{i \in I(\bar{t})} R_i \\ \text{Modif-Inv}(\bar{t}) & = \{x \in X \mid \llbracket \bigwedge_{i \in I(\bar{t})} Inv_i(\ell_i) \rrbracket_x \neq \llbracket \bigwedge_{i \in I(\bar{t})} Inv_i(\ell'_i) \rrbracket_x\} \\ \text{Testé}(\bar{t}) & = X_{inv} \cup \text{Horloges}(g(\bar{t})) \\ \text{Modifié}(\bar{t}) & = R(\bar{t}) \cup \text{Modif-Inv}(\bar{t}) \end{cases}$$

Intuitivement,  $\text{Eti}(\bar{t})$ ,  $g(\bar{t})$  et  $R(\bar{t})$  représentent respectivement l'étiquette, la garde et la remise à zéro associées à la transition synchronisée.  $\text{Modif-Inv}(\bar{t})$  représente l'ensemble des horloges pour

<sup>4</sup>Nous ne considérons pas d'ensembles d'états de contrôle finals et répétés dans ce contexte.

<sup>5</sup>L'opérateur *Horloges* qui permet d'obtenir l'ensemble des horloges impliquées dans la définition d'une contrainte est défini page 30.

<sup>6</sup>L'opération  $\llbracket \varphi \rrbracket_x$  consistant à projeter  $\llbracket \varphi \rrbracket$  sur l'horloge  $x$  a été définie page 30.

lesquelles le franchissement de  $\bar{t}$  modifie la contrainte d'invariant globale.  $Testé(\bar{t})$  dénote l'ensemble des horloges apparaissant dans un invariant ou testées par la garde de la synchronisation  $\bar{t}$ .  $Modifié(\bar{t})$  dénote l'ensemble des horloges dont la valeur ou la contrainte d'invariant est modifiée par le franchissement de  $\bar{t}$ . Ceci peut être dû à une remise à zéro de l'horloge ou à la modification de la contrainte d'invariant globale portant sur cette horloge. Les définitions des ensembles  $Cons(\bar{t})$ ,  $Prod(\bar{t})$  et  $Lu(\bar{t})$  tiennent compte de ces ensembles. Notons qu'une horloge dont l'invariant est modifié est consommée puis reproduite.

$$\begin{cases} Cons(\bar{t}) &= (\cup_{i \in I(\bar{t})} \ell_i) \cup Modifié(\bar{t}) \\ Prod(\bar{t}) &= (\cup_{i \in I(\bar{t})} \ell'_i) \cup Modifié(\bar{t}) \\ Lu(\bar{t}) &= Testé(\bar{t}) \setminus Modifié(\bar{t}) \end{cases}$$

Nous définissons un processus branchant d'un réseau d'automates temporisés comme un réseau d'occurrences contextuel muni d'une fonction d'étiquetage obtenue à partir des précédentes définitions.

**Définition 7.8** (Processus branchant d'un réseau d'automates temporisés). *Soit  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$  un réseau d'automates temporisés. Un processus branchant de  $\mathcal{A}$  est une paire  $\beta = (\mathcal{N}, \mu)$  constituée d'un réseau d'occurrences contextuel  $\mathcal{N} = (P, T, \Sigma_\varepsilon, \bullet(\cdot), (\cdot)^\bullet, \circ(\cdot), M_0, \Lambda)$  et d'une application  $\mu$  de  $P \cup T$  dans  $L \cup X \cup Sync$  vérifiant les conditions suivantes :*

- $\mu(P) \subseteq L \cup X$ ,
- $\mu(T) \subseteq Sync$ ,
- $\mu$  est une bijection entre  $M_0$  et  $\{\ell_{i,0} \mid 1 \leq i \leq n\} \cup X$ ,
- pour tout  $t \in T$ ,  $\mu$  est une bijection entre les ensembles suivants :

$$\begin{cases} \bullet t & \text{est envoyé sur } Cons(\mu(t)) \\ t^\bullet & \text{est envoyé sur } Prod(\mu(t)) \\ \circ t & \text{est envoyé sur } Lu(\mu(t)) \end{cases}$$

- il n'y a pas de redondance, i.e. pour deux événements  $t, t' \in T$ , si  $\mu(t) = \mu(t')$ ,  $\bullet t = \bullet t'$  et  $\circ t = \circ t'$ , alors  $t = t'$ ,
- l'étiquetage de  $T$  est cohérent : pour tout  $t \in T$ , nous avons  $\Lambda(t) = Eti(\mu(t))$ .

Les conditions représentent ou bien des états de contrôle (éléments de  $L$ ) ou bien des horloges (éléments de  $X$ ) et les événements représentent les transitions synchronisées du réseau. D'autre part, les événements sont connectés aux conditions selon les définitions des ensembles  $Cons(\mu(t))$ ,  $Prod(\mu(t))$  et  $Lu(\mu(t))$ . Dans ces définitions, le rôle des états de contrôle est resté inchangé. Pour les horloges, leur rôle correspond à la description faite plus haut de ces ensembles :

- une horloge remise à zéro doit être consommée et reproduite,
- une horloge apparaissant dans un invariant doit apparaître ou bien en lecture, ou bien en consommation,
- une horloge intervenant uniquement dans la garde doit apparaître en lecture.

Dans [VSY98, Win02], une relation de préfixe est définie entre les processus branchants d'un réseau de Petri avec arcs de lecture. Cette définition s'étend au cadre des processus branchants d'un réseau d'automates temporisés  $\mathcal{A}$ . On démontre alors que l'ensemble de ces processus forme par rapport à cette relation de préfixe un treillis complet. En particulier, ceci implique l'existence d'un préfixe maximal pour cette relation. C'est ce préfixe qui est appelé dans la suite *le dépliage non temporisé du réseau d'automates temporisés  $\mathcal{A}$* .

**Exemple 7.9** (Exemples 7.3 et 7.4 poursuivis). Les dépliages non temporisés (ou seulement un préfixe) des réseaux d'automates temporisés  $\mathcal{B}$  et  $\mathcal{C}$  sont représentés sur la figure 7.5.

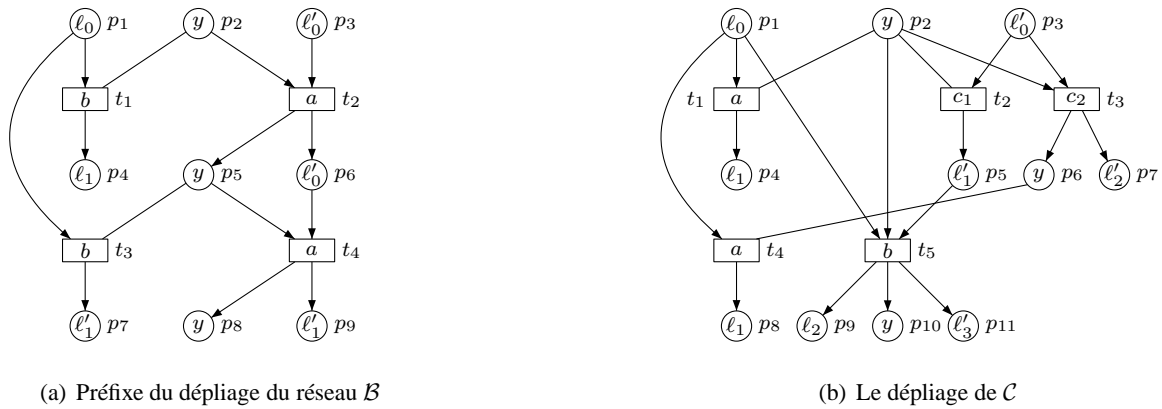


FIG. 7.5 – Deux dépliages non temporisés.

Considérons d'abord le réseau  $\mathcal{B}$ . Notons l'ajout d'une place étiquetée  $y$  représentant l'horloge  $y$ . D'après les définitions précédentes, cette place est en lecture de tout événement étiqueté  $b$  et est consommée puis reproduite par tout événement étiqueté  $a$ . En particulier, ceci conduit, après ajout de l'événement  $t_2$ , à une nouvelle occurrence d'un événement étiqueté  $b$ , à savoir l'événement  $t_3$ , qui n'existait pas dans le dépliage du réseau d'automates finis sous-jacent. Nous avons ainsi obtenu ces occurrences supplémentaires de  $b$  qui manquaient précédemment.

Considérons à présent le réseau  $\mathcal{C}$ . À nouveau, une place supplémentaire est présente, étiquetée  $y$ , afin de représenter l'horloge  $y$ . Selon les définitions précédentes la transition  $t_2$  lit la place  $p_2$  tandis que la transition  $t_3$  la consomme et reproduit une nouvelle place étiquetée  $y$ , la place  $p_6$ . En effet, le tir de  $t_2$ , étiquetée  $c_1$ , ne modifie pas l'invariant sur  $y$  contrairement au tir de  $t_3$ . En particulier, le tir de  $t_3$  crée donc une nouvelle place étiquetée par  $y$ . Ceci donne lieu à une nouvelle occurrence d'une transition étiquetée  $a$ , à savoir  $t_4$ . Ceci est cohérent avec les remarques faites plus haut : nous obtenons deux occurrences de  $a$ ,  $t_1$  correspond au franchissement sous l'invariant  $y \leq 1$  tandis que  $t_4$  correspond au franchissement sous l'invariant tt. Par la suite, nous identifierons de plus que les contraintes temporelles impliquent que le tir de  $t_1$  est impossible (invariant  $y \leq 1$  et garde  $y \geq 3$ ).

Nous avons donc vu pour chacun de ces deux exemples comment les places étiquetées par des horloges et les nouvelles définitions des ensembles  $Cons(\bar{t})$ ,  $Prod(\mu(t))$  et  $Lu(\mu(t))$  intégrant des arcs de lecture permettent de produire de nouvelles occurrences des transitions. Notons également que les arcs de lecture permettent un « partage » des places et augmentent ainsi la taille de la relation de concurrence. Si nous avons simplement modélisé les horloges par des places dédiées mais utilisé des arcs de consommation et de production, nous obtiendrions davantage d'entrelacements du système. Sur notre premier exemple, si  $b$  reproduisait  $y$ , nous devrions franchir à nouveau  $a$  à l'issue du tir de  $b$ . Sur le second exemple, les transitions  $t_1$  et  $t_2$  ont toutes deux dans leur ensemble de place en lecture la place  $p_2$ . Ceci permet de ne pas les ordonner par la relation  $<$  ni même par la relation  $\prec$ . Elles sont concurrentes vis-à-vis de  $<$  et  $\prec$ . Cette notion sera formalisée par la suite.  $\lrcorner$

**Processus non branchant d'un réseau d'automates temporisés.** Afin de raisonner sur un processus branchant d'un réseau d'automates temporisés, nous introduisons un ensemble de notions liées au réseau d'occurrences contextuel dont il est issu. Celles-ci permettent en particulier de définir la notion de processus non branchant qui correspond à la représentation des véritables exécutions du réseau.

**Définition 7.10** (Processus non branchant, configuration, coupe, passé causal). Soit  $\beta = (\mathcal{N}, \mu)$  un processus branchant d'un réseau d'automates temporisés  $\mathcal{A}$ . Nous notons  $T$  (respectivement  $P$ ) l'ensemble des événements (respectivement des conditions) de  $\mathcal{N}$ .

Un sous-ensemble  $C \subseteq T$  est une configuration s'il satisfait les conditions suivantes :

- $\forall t' \in C, \forall t \in T, t \leq t' \Rightarrow t \in C$  ( $C$  est clos par le bas pour  $\leq$ ),
- la relation  $\prec$  restreinte à  $C$  est un ordre partiel,
- $\forall t, t' \in C, \neg(t \# t')$  ( $C$  est sans conflit).

Nous considérons le réseau d'occurrences contextuel  $\mathcal{N}'$  associé à une restriction  $(P', T') \subseteq (P, T)$  de  $\mathcal{N}$ . Nous notons  $\mu'$  la restriction de  $\mu$  à  $\mathcal{N}'$ . Alors  $\beta' = (\mathcal{N}', \mu')$  est appelé un processus non branchant de  $\beta$  s'il satisfait les cinq conditions suivantes :

- $\forall t \in T, \forall p \in \bullet t \cup \circ t \cup t^\bullet, t \in T' \Rightarrow p \in P'$  (les événements sont cohérents avec  $\beta$ ),
- $\forall p \in P, \forall t \in \bullet p, p \in P' \Rightarrow t \in T'$  (les conditions sont cohérentes avec  $\beta$ ),
- la relation  $\prec$  restreinte à  $P' \cup T'$  est un ordre partiel,
- $\forall x, y \in P' \cup T', \neg(x \# y)$  ( $\mathcal{N}'$  est sans conflit),
- $\text{Min}(P') = \text{Min}(P)$ .

Remarquons que si  $\beta'$  est un processus non branchant de  $\beta$ , alors l'ensemble des événements de  $\beta'$  constitue une configuration notée  $[\beta']$ . Réciproquement, étant donnée une configuration  $C$ , il existe un unique processus non branchant, noté  $\beta_C$ , tel que  $C = [\beta_C]$ .

Étant donnés deux nœuds  $x$  et  $y$  de  $\beta'$ , nous disons que  $x$  et  $y$  sont :

- faiblement concurrents, noté  $x \text{ co} < y$ , si et seulement si ni  $x \leq y$  ni  $y \leq x$  n'est vérifié,
- fortement concurrents, noté  $x \text{ co} \prec y$ , si et seulement si ni  $x \preceq y$  ni  $y \preceq x$  n'est vérifié.

Un ensemble  $Y$  de nœuds de  $\beta'$  est un  $\text{co} <$ -ensemble (respectivement un  $\text{co} \prec$ -ensemble) si les éléments de  $Y$  sont deux à deux faiblement concurrents (respectivement deux à deux fortement concurrents). Une coupe est un  $\text{co} \prec$ -ensemble maximal constitué uniquement de conditions.

Si  $C$  est une configuration, nous associons à  $C$  la coupe notée  $\text{Cut}(C)$  définie par  $\text{Cut}(C) = (\text{Min}(P) \cup C^\bullet) \setminus \bullet C$ . Étant donné un processus non branchant  $\beta'$ , nous noterons simplement  $\text{Cut}(\beta')$  la coupe  $\text{Cut}([\beta'])$ .

Étant donné un processus non branchant  $\beta'$  de  $\beta$  et un événement  $t$  appartenant à  $\beta'$ , nous définissons le passé causal fort de  $t$  par rapport à  $\beta'$ , noté  $[t]_{\beta'}$ , comme l'ensemble  $[t]_{\beta'} = \{t' \in T' \mid t' \preceq t\}$ . Le passé causal faible de  $t$ , noté  $[t]$ , est défini par  $[t] = \{t' \in T' \mid t' < t\}$ . Il est facile de vérifier que  $[t]$  est une configuration qui ne dépend pas du choix de  $\beta'$  et qui constitue de plus son passé causal fort minimal, i.e. nous avons l'identité  $[t] = \bigcap_{\beta'} [t]_{\beta'}$  où  $\beta'$  décrit l'ensemble des processus non branchants de  $\beta$  contenant  $t$ . Nous appellerons donc également passé causal minimal de  $t$  son passé causal faible.

Enfin, étant donné deux configurations  $C_1$  et  $C_2$  de  $\beta$ , nous dirons que la configuration  $C_2$  étend la configuration  $C_1$ , noté  $C_1 \sqsubseteq C_2$ , si et seulement si  $C_1 \subseteq C_2$  et si étant donné deux événements  $t_1$  et  $t_2$  appartenant respectivement à  $C_1$  et  $C_2 \setminus \beta_1$ , nous n'avons pas  $t_2 \prec t_1$  dans  $C_2$ . Posant  $Y = C_2 \setminus C_1$ , nous écrirons alors  $C_2 = C_1 \oplus Y$ . Ces définitions s'étendent naturellement aux processus non branchants en les appliquant à leurs configurations. Nous étendons également la notation  $\sqsubseteq$ .

**Remarque 7.11** (À propos des définitions portant sur les processus non branchants).

- La relation  $\prec$  dépend du processus non branchant dans lequel elle est calculée. En effet, ajouter des événements peut créer des dépendances supplémentaires et donc des ordonnancements supplémentaires. Pour le réseau d'occurrences contextuel de la figure 7.4(b) page 192, la relation  $\prec$  considérée sur l'ensemble du réseau (qui correspond à un processus non branchant) donne  $t_1 < t_3$  et  $t_3 \prec t_2$  d'où  $t_1 \prec t_2$ , tandis que dans le processus non branchant correspondant uniquement au sous-ensemble d'événements  $\{t_1, t_2\}$ , les événements  $t_1$  et  $t_2$  ne sont pas ordonnés pour  $\prec$ .
- Le passé causal minimal d'un événement  $t$  peut être calculé de façon itérative par  $[t] = \{t\} \cup \bigcup_{t' \in \bullet(\bullet t \cup \circ t)} [t']$ . Grâce à la structure de treillis des processus branchants d'un réseau d'automates temporisés,  $[t]$  ne dépend pas de  $\beta$ .

┘

Nous illustrons ces définitions sur les exemples précédents.

**Exemple 7.12** (Exemple 7.4 poursuivi). Considérons le processus branchant  $\beta$  associé au réseau d'automates temporisés  $\mathcal{C}$  représenté sur la sous-figure 7.5(b). Alors le sous-graphe constitué des nœuds  $\{p_1, p_2, p_3, p_4, p_6, p_7\} \cup \{t_1, t_3\}$  est un processus non branchant, disons  $\beta'$ . Ce processus non branchant comporte une occurrence de  $a$  et une occurrence de  $c_2$ . D'autre part, la configuration associée à  $\beta'$  est  $\{t_1, t_3\}$ . Nous avons  $[t_3]_{\beta'} = \{t_1, t_3\}$ . En effet, dans  $\beta'$ , afin de franchir l'événement  $t_3$ , nous devons franchir avant l'événement  $t_1$ , i.e.  $t_1 \prec t_3$ . Si nous considérons le passé causal minimal  $[t_3]$  de  $t_3$ , alors l'événement  $t_1$  n'est plus présent ( $t_1 \not\prec t_3$ ) et nous obtenons  $[t_3] = \{t_3\}$ .

D'autre part, notons  $\beta_1$  et  $\beta_3$  les processus non branchants associés respectivement aux événements  $t_1$  et  $t_3$ , i.e. de configurations respectives  $\{t_1\}$  et  $\{t_3\}$ . Nous avons les deux inclusions  $[\beta_i] \subseteq [\beta']$  pour  $i \in \{1, 3\}$ . Pourtant, le processus  $\beta'$  n'étend pas chacun de ces deux processus. En effet, du fait de la relation  $t_1 \prec t_3$ , nous obtenons  $\beta_1 \sqsubseteq \beta'$  mais  $\beta_3 \not\sqsubseteq \beta'$ . ┘

Avant de présenter la construction du dépliage non temporisé, nous établissons deux lemmes techniques dépendant uniquement de notre définition des processus branchants d'un automate temporisé. Ces deux lemmes seront utiles par la suite. Le premier établit une propriété classique [ER99] des coupes dans un dépliage d'automates qui montre que celles-ci contiennent toute l'information sur l'état global du réseau.

**Lemme 7.13** (Propriétés des coupes). *Soit  $\mathcal{A}$  un réseau d'automates temporisés,  $\beta = (\mathcal{N}, \mu)$  un processus branchant de  $\mathcal{A}$  et  $\mathcal{C}$  une coupe de  $\beta$ . Alors  $\mathcal{C}$  vérifie les deux propriétés suivantes :*

- l'ensemble  $\mathcal{C} \cap \mu^{-1}(X)$  est en bijection (par  $\mu$ ) avec l'ensemble  $X$ ,
- l'image par  $\mu$  de l'ensemble  $\mathcal{C} \cap \mu^{-1}(L)$  contient exactement un état de contrôle par automate du réseau.

Étant donnée une horloge  $x \in X$ , nous notons  $p_x(\mathcal{C})$  l'unique condition appartenant à  $\mathcal{C}$  dont l'image par  $\mu$  est  $x$ . De même, nous notons  $\ell(\mathcal{C})$  l'unique vecteur élément de  $\prod_{1 \leq i \leq n} L_i$  image par  $\mu$  de  $\mathcal{C} \cap \mu^{-1}(L)$ .

**Preuve.** La coupe initiale vérifie ces deux propriétés. Il suffit de constater qu'elles sont stables par ajout d'un événement : d'après les définitions de  $Cons(\bar{t})$  et  $Prod(\bar{t})$  pour une transition du réseau d'automates temporisés  $\bar{t}$ , à chaque fois qu'une place correspondant à une horloge est consommée, une place correspondant à la même horloge est également produite. De même, à chaque fois qu'une place correspondant à un état de contrôle d'un automate du réseau, une nouvelle place dont l'étiquette est un état de contrôle du même automate est produite par la transition. □

Ce second lemme exprime que l'ensemble des conditions consommées ou lues par un événement du dépliage est suffisant pour déterminer l'invariant courant lors du franchissement de cet événement. Il formalise la notion de localité des événements annoncée dans l'introduction au sens où l'état local associé aux places en entrée d'une transition détermine l'invariant de l'état global, pour tout état global étendant cet état local.

**Lemme 7.14** (Localité). *Soit  $\mathcal{A}$  un réseau d'automates temporisés et  $\beta$  un processus branchant de  $\mathcal{A}$ . Considérons un événement  $t$  de  $\beta$ . Alors étant données deux coupes  $\mathcal{C}$  et  $\mathcal{C}'$  vérifiant  $\bullet t \cup \circ t \subseteq \mathcal{C}$  et  $\bullet t \cup \circ t \subseteq \mathcal{C}'$ , nous avons l'identité suivante :*

$$\llbracket \text{Inv}(\ell(\mathcal{C})) \rrbracket = \llbracket \text{Inv}(\ell(\mathcal{C}')) \rrbracket$$

**Preuve.** Toute horloge  $x \in X$  apparaissant dans un invariant de  $\mathcal{A}$  (i.e.  $x \in X_{inv}$ ) est représentée dans  $\bullet t \cup \circ t$  d'après les définitions de  $\text{Cons}(\bar{t})$  et de  $\text{Lu}(\bar{t})$  de la page 193. De plus, en examinant ces mêmes définitions, il est facile de vérifier que lors du franchissement d'une transition, si la contrainte d'invariant portant sur une horloge ou la valeur de l'horloge ont été modifiées, alors la condition correspondant à cette horloge est consommée et une nouvelle condition est produite. Ainsi, deux coupes étendant l'ensemble  $\bullet t \cup \circ t$  contiennent nécessairement les mêmes contraintes d'invariants et les mêmes valeurs d'horloges pour toutes les horloges d'invariant ( $X_{inv}$ ), ce qui démontre le résultat.  $\square$

**Construction du dépliage.** Nous adoptons les notations introduites dans [VSY98] afin de présenter un semi-algorithme pour la construction du dépliage non temporisé d'un réseau d'automates temporisés  $\mathcal{A}$ . Dans cet algorithme, une condition est codée comme une paire  $(p, t)$  où  $p$  est l'étiquette de cette condition et  $t$  est l'unique événement ayant produit cette condition ( $t$  est égal à l'ensemble vide si cette condition n'est produite par aucun événement – cas des conditions minimales). Un événement est représenté par un triplet  $(\bar{t}, Y_c, Y_l)$  où  $\bar{t} \in \text{Sync}$  est l'étiquette de  $t$  et où  $Y_c$  et  $Y_l$  sont deux listes de pointeurs vers des conditions. Intuitivement,  $Y_c$  représente l'ensemble des conditions consommées par  $t$  tandis que  $Y_l$  représente l'ensemble des conditions lues par  $t$ .

**Définition 7.15** (Extensions Possibles (EP)). *Soit  $\beta = (\mathcal{N}, \mu)$  un processus branchant d'un réseau d'automates temporisés  $\mathcal{A}$ . Les extensions possibles de  $\beta$  sont les triplets  $t = (\bar{t}, Y_c, Y_l)$  où  $\bar{t}$  est un élément de  $\text{Sync}$  tel qu'il existe un processus non branchant  $\beta'$  vérifiant les conditions suivantes :*

- $Y_c \cup Y_l$  est un  $\text{co } \prec$ -ensemble de conditions de  $\beta'$ ,
- la restriction de  $\mu$  à  $Y_c$  (respectivement à  $Y_l$ ) est une bijection vers  $\text{Cons}(\bar{t})$  (respectivement vers  $\text{Lu}(\bar{t})$ ),
- $(\bar{t}, Y_c, Y_l)$  n'appartient pas à  $\beta$ .

Dans ce cas, nous définissons l'extension de  $\beta$  par  $t$ , notée  $\text{Extension}(\beta, t)$ , comme le processus branchant  $\beta'$  obtenu à partir de  $\beta$  en ajoutant un événement  $e$  étiqueté par  $\bar{t}$  et un ensemble  $Z$  de nouvelles conditions en bijection (par  $\mu$ ) avec  $\text{Prod}(\bar{e})$ . L'événement  $e$  est connecté par des arcs de consommation aux places de  $Y_c$ , des arcs de lecture aux places de  $Y_l$  et des arcs de production aux nouvelles places de  $Z$ .

Le théorème suivant établit la correction de l'algorithme 5.

**Théorème 7.16.** *Soit  $\mathcal{A}$  un réseau d'automates temporisés. L'algorithme 5 est un semi-algorithme permettant de calculer le dépliage non temporisé  $\text{Dépliage}(\mathcal{A})$  de  $\mathcal{A}$ .*

**Algorithme 5** Construction du dépliage non temporisé (semi-algorithme)**Entrée :** Un réseau d'automates temporisés  $\mathcal{A}$ .**Sortie :** Le dépliage non temporisé  $Dépliage(\mathcal{A})$  de  $\mathcal{A}$ .

- 1:  $Dépliage := \{(\ell_{1,0}, \emptyset), \dots, (\ell_{n,0}, \emptyset)\} \cup \{(x, \emptyset) \mid x \in X\};$  /\* Initialisation \*/
- 2:  $ep := EP(Dépliage);$  /\* Extensions possibles \*/
- 3: **Tant Que**  $ep \neq \emptyset$  **Faire**
- 4:   Choisir un événement  $t = (\bar{t}, Y_c, Y_l)$  dans  $ep$ . /\*  $\bar{t}$  est l'étiquette de  $t$  \*/
- 5:    $Dépliage := Extension(Dépliage, t);$  /\* Extension du processus \*/
- 6:    $ep := EP(Dépliage);$  /\* Mise à jour de l'ensemble  $ep$  \*/
- 7: **Fin Tant Que**
- 8: **Retourner**  $Dépliage$  ;

## 7.4 Dépliage temporisé

Nous avons décrit dans la section précédente une structure causale permettant d'introduire les aspects temporels au niveau discret en dupliquant certains événements lorsque les conditions de franchissement d'un événement diffèrent. Il s'agit à présent de définir une représentation symbolique des contraintes temporisées attachées à un processus non branchant de sorte à décrire de façon symbolique l'ensemble des exécutions temporisées du réseau d'automates temporisés. Ceci va également permettre de limiter la construction du dépliage en détectant des événements dont la partie temporisée n'est pas valide, comme c'est le cas pour l'événement  $t_1$  du dépliage du réseau d'automates temporisés  $\mathcal{C}$  (voir exemple 7.4).

### 7.4.1 Processus non branchant temporisé

**Exécutions temporisées d'un réseau d'automates temporisés.** Rappelons d'abord le comportement d'un réseau d'automates temporisés  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$ , où nous notons  $\mathcal{A}_i = (\Sigma, X_i, L_i, T_i, Inv_i, \ell_{i,0})$  pour tout  $i$ . Une configuration d'un tel réseau est une paire  $(\ell, v)$  dans laquelle  $\ell \in \prod_{1 \leq i \leq n} L_i$  et  $v \in \mathbb{T}^X$ , avec  $X = \cup_{1 \leq i \leq n} X_i$ . Une *exécution temporisée finie* de  $\mathcal{A}$  s'écrit alors :

$$\rho = (\ell_0, v_0) \xrightarrow{\bar{t}_1, d_1} (\ell_1, v_1) \xrightarrow{\bar{t}_2, d_2} \dots \xrightarrow{\bar{t}_p, d_p} (\ell_p, v_p)$$

où pour tout  $i$ ,  $\bar{t}_i \in Sync$  et  $d_i \in \mathbb{T}$  vérifiant de plus  $d_i \leq d_{i+1}$ . Intuitivement,  $d_i$  indique la date à laquelle la transition synchronisée du réseau  $\bar{t}_i$  a lieu. Notons que nous ne considérons que des exécutions qui peuvent *réellement* survenir dans  $\mathcal{A}$ . Nous associons à cette exécution la *séquence temporisée*  $\sigma = (\bar{t}_i, d_i)_{1 \leq i \leq p}$ , mot fini sur l'alphabet  $Sync \times \mathbb{T}$  dont la projection sur la seconde composante forme une séquence finie et croissante de dates. Réciproquement, à une séquence temporisée est associée une unique exécution temporisée dans  $\mathcal{A}$ . À nouveau, par séquence temporisée, nous sous-entendons une séquence correspondant à une *exécution réelle* du système.

Dans un premier temps, nous attachons des dates, *i.e.* des éléments de  $\mathbb{T}$ , aux nœuds d'un processus non branchant  $\beta = (\mathcal{N}, \mu)$ , définissant ainsi un *processus non branchant temporisé*. Avant de donner la définition formelle de ce nouvel objet, nous donnons l'intuition de ces dates. Implicitement, nous considérons une exécution temporisée du réseau donnant lieu à ce processus non branchant. Nous formaliserons cette idée ensuite. D'abord, nous associons une date  $\mathbf{d}_f(t)$  à chaque événement  $t$  du processus non branchant. Cette date représente la date absolue à laquelle l'événement a été franchi. Ensuite, étant donnée une condition  $p$  de  $\beta$ , nous distinguons deux cas, selon que l'image de  $p$  par

$\mu$  est un état de contrôle ou une horloge. Dans le premier cas, nous associons à  $p$  la date  $\mathbf{d}_p(p)$ , représentant la date absolue à laquelle la condition a été produite. Dans le second cas, nous associons à  $p$  les trois dates  $\mathbf{d}_p(p)$ ,  $\mathbf{d}_c(p)$  et  $\mathbf{d}_r(p)$ . La première correspond comme précédemment à la date absolue à laquelle la condition a été produite. La seconde correspond à la date absolue à laquelle la condition a été consommée. Si ce n'est pas le cas, alors cette date vaut  $+\infty$ . Enfin, la dernière date  $\mathbf{d}_r(p)$  représente la date absolue à laquelle l'horloge  $\mu(p)$  a été remise à zéro pour la dernière fois.

**Définition 7.17** (Processus non branchant temporisé). *Soit  $\beta$  un processus non branchant d'un réseau d'automates temporisés  $\mathcal{A}$ , d'ensemble d'événements  $T$  et de conditions  $P$ . Nous définissons l'ensemble des horloges que nous allons considérer à l'aide de l'application  $\text{Var}$  suivante :*

$$\begin{aligned} \forall t \in T, \quad \text{Var}(t) &= \{\mathbf{d}_f(t)\} \\ \forall p \in P \cap \mu^{-1}(L), \quad \text{Var}(p) &= \{\mathbf{d}_p(p)\} \\ \forall p \in P \cap \mu^{-1}(X), \quad \text{Var}(p) &= \{\mathbf{d}_p(p), \mathbf{d}_c(p), \mathbf{d}_r(p)\} \end{aligned}$$

Nous définissons alors l'ensemble des variables d'horloges du processus non branchant  $\beta$ , noté  $V(\beta)$ , par  $V(\beta) = \text{Var}(P \cup T)$ . Une valuation de  $\beta$  est une valuation sur l'ensemble  $V(\beta)$ , i.e. un élément de  $\mathbb{T}^{V(\beta)}$ . Un processus non branchant temporisé est alors défini comme une paire  $(\beta, v)$  constituée d'un processus non branchant  $\beta$  et d'une valuation  $v$  de  $\beta$ .

Afin d'illustrer cette définition, commençons par décrire la construction permettant d'associer un processus non branchant temporisé à une séquence temporisée. Cette définition est conforme à l'intuition donnée précédemment pour les différentes dates attachées à un processus non branchant.

**Définition 7.18** (Processus non branchant temporisé associé à une séquence temporisée). *Soit  $\mathcal{A}$  un réseau d'automates temporisés et  $\sigma$  une séquence temporisée de  $\mathcal{A}$ . Nous nous plaçons dans le cadre du dépliage non temporisé de  $\mathcal{A}$ . Le processus non branchant temporisé associé à  $\sigma$ , noté  $\text{Pr-T}(\sigma)$ , est défini par induction ainsi :*

- Si  $\sigma$  est la séquence vide  $\varepsilon$ , alors  $\text{Pr-T}(\varepsilon) = (\beta, v)$  est défini comme la paire du processus non branchant  $\beta$  réduit à l'ensemble de conditions  $\text{Min}(P)$  et de la valuation  $v$  définie par :
  - $\forall p \in \text{Min}(P), v(\mathbf{d}_p(p)) = 0$ ,
  - $\forall p \in \text{Min}(P) \cap \mu^{-1}(X), v(\mathbf{d}_r(p)) = 0$  et  $v(\mathbf{d}_c(p))$  est choisie de façon non déterministe dans l'intervalle  $[0, +\infty[$ .
- Si  $\sigma = \sigma'(\bar{t}, d)$  (rappelons que  $d$  représente la date absolue de l'occurrence de  $\bar{t}$  dans  $\sigma$ ), nous allons définir  $\text{Pr-T}(\sigma) = (\beta, v)$ . Notons  $\text{Pr-T}(\sigma') = (\beta', v')$  le processus non branchant temporisé associé à  $\sigma'$ . Écrivons de plus  $\mathcal{C} = \text{Cut}(\beta')$  la configuration de  $\beta'$ . Il existe alors une unique extension  $\text{Extension}(\beta', t)$  de  $\beta'$  par un triplet  $t = (\bar{t}, Y_c, Y_l)$  tel que  $Y_c \cup Y_l \subseteq \mathcal{C}$ . Nous définissons  $\beta$  comme étant cette extension  $\text{Extension}(\beta', t)$ . La valuation  $v$  de  $\beta$  est quant à elle définie ainsi :
  - $v$  préserve les valeurs de  $v'$  sur les conditions et les transitions de  $\beta'$ , excepté les conditions  $p \in \bullet t \cap \mu^{-1}(X)$  pour lesquelles nous définissons  $v(\mathbf{d}_c(p)) = d$ ,
  - $v(\mathbf{d}_f(t)) = d$ ,
  - $\forall p \in t^\bullet, v(\mathbf{d}_p(p)) = d$ ,
  - $\forall p \in t^\bullet \cap \mu^{-1}(X), v(\mathbf{d}_c(p))$  est choisie de façon non déterministe dans l'intervalle  $[d, +\infty[$ ,
  - $\forall p \in t^\bullet \cap \mu^{-1}(X)$ , si  $\mu(p) \in R(\bar{t})$ , alors  $v(\mathbf{d}_r(p)) = d$ , sinon  $v(\mathbf{d}_r(p)) = v(\mathbf{d}_r(p'))$  où  $p'$  est l'unique condition de  $\mathcal{C}$  étiquetée par  $\mu(p)$ .

Notons que dans la définition précédente, certaines valeurs d'horloges sont choisies de façon non déterministe. Ceci concerne les horloges  $\mathbf{d}_c(p)$  où  $p$  est une condition représentant une horloge de la



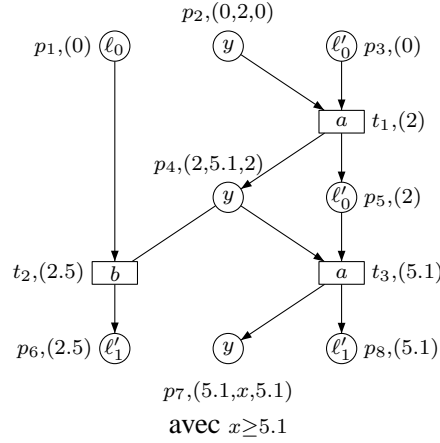


FIG. 7.6 – Le processus non branchant temporel associé à une exécution temporel.

coupe du processus non branchant. En particulier, ces conditions n'ont « pas encore » été consommées, il n'est donc pas possible de donner une valeur à ces horloges. Cependant, leur consommation ne peut survenir que dans le futur et nous choisissons donc une valeur vérifiant cette contrainte.

**Définition 7.19** (Configuration atteinte dans le réseau d'automates temporels). Soit  $\mathcal{A}$  un réseau d'automates temporels. Étant donné un processus non branchant temporel  $(\beta, v)$ , nous définissons la configuration  $\text{Config}(\beta, v)$  du réseau d'automates temporels comme la paire  $(\ell, \nu)$  où :

$$\begin{cases} \ell = \ell(\text{Cut}(\beta)) \\ \forall x \in X, \nu(x) = v(\mathbf{d}_p(p_x)) - v(\mathbf{d}_r(p_x)) \end{cases}$$

où  $p_x = p_x(\text{Cut}(\beta))$  est l'unique condition de  $\text{Cut}(\beta)$  étiquetée par  $x$  (voir le lemme 7.13).

Le lemme suivant, dont la preuve s'obtient par induction sur la longueur de la séquence temporel, énonce que la définition précédente est cohérente pour les processus non branchants temporels obtenus à partir de séquences temporels.

**Lemme 7.20.** Soit  $\mathcal{A}$  un réseau d'automates temporels et  $\rho$  une exécution temporel de  $\mathcal{A}$  :

$$\rho = (\ell_0, v_0) \xrightarrow{\bar{t}_1, d_1} (\ell_1, v_1) \xrightarrow{\bar{t}_2, d_2} \dots \xrightarrow{\bar{t}_p, d_p} (\ell_p, v_p)$$

Notons  $\sigma$  la séquence temporel associée à  $\rho$ . Nous avons alors l'égalité suivante :

$$(\ell_p, v_p) = \text{Config}(\text{Pr-T}(\sigma))$$

**Exemple 7.21** (Exemple 7.3 poursuivi encore). Nous considérons à nouveau le réseau d'automates temporels  $\mathcal{B}$  représenté sur la sous-figure 7.1(a). La séquence temporel  $\sigma = (a, 2)(b, 2.5)(a, 5.1)$  est acceptée par  $\mathcal{B}$ . Le processus non branchant temporel  $\text{Pr-T}(\sigma)$  est représenté sur la figure 7.6. La valuation est représentée sur le graphique : les dates associées aux nœuds sont indiquées à côté de ceux-ci. Pour les conditions correspondant à des horloges, le triplet indiqué correspond aux trois variables d'horloges suivantes dans cet ordre :  $(\mathbf{d}_p, \mathbf{d}_c, \mathbf{d}_r)$ .  $\lrcorner$

### 7.4.2 Cohérence temporelle d'un processus non branchant temporisé

Une fois ces définitions introduites, notre objectif est de caractériser l'ensemble des processus branchants temporisés correspondant à des séquences temporisés. Pour cela, nous utilisons des zones <sup>7</sup> qui permettent de représenter des ensembles infinis de valuations. Ces zones vont être définies localement, au niveau de chaque événement du processus non branchant.

Nous introduisons d'abord quelques notations. Soit  $t$  un événement du dépliage  $Dépliage(\mathcal{A})$  d'un réseau d'automates temporisés  $\mathcal{A}$ , nous définissons la coupe  $\mathcal{C}^+ = Cut([t])$ . Cette coupe représente intuitivement l'état global de  $\mathcal{A}$  à l'issue du franchissement de l'ensemble des événements de la configuration  $[t]$ . De plus, il est facile de vérifier que l'ensemble  $[t] \setminus \{t\}$  forme une configuration. Nous notons  $\mathcal{C}^-$  la coupe qui lui est associée. Intuitivement, celle-ci représente l'état du système avant le franchissement de  $t$ . D'après le lemme 7.13, chacune de ces deux coupes est formée, via la fonction d'étiquetage  $\mu$ , d'une condition correspondant à un état de contrôle pour chaque automate du réseau et d'une condition par horloge du réseau. Le vecteur d'états de contrôle associé à  $\mathcal{C}^-$  est noté  $\ell^-$ . D'autre part, à nouveau d'après le lemme 7.13, étant donnée une horloge  $x \in X$ , il existe dans la coupe  $\mathcal{C}^-$  (respectivement dans la coupe  $\mathcal{C}^+$ ) une unique condition, notée  $p_x^-$  (respectivement  $p_x^+$ ), dont l'image par  $\mu$  vaut  $x$ . Nous définissons alors les deux termes suivants :

$$\begin{cases} v_t^-(x) &= \mathbf{d}_f(t) - \mathbf{d}_r(p_x^-) \\ v_t^+(x) &= \mathbf{d}_f(t) - \mathbf{d}_r(p_x^+) \end{cases}$$

Le terme  $v_t^-(x)$  représente la valeur de l'horloge  $x$  dans le réseau  $\mathcal{A}$  lors du franchissement de  $t$ , tandis que le terme  $v_t^+(x)$  représente sa valeur à l'issue du franchissement de  $t$ . En effet, la valeur de l'horloge s'obtient en calculant la différence entre la date courante et la date de dernière remise à zéro de l'horloge.

**Définition 7.22** (Cohérence temporelle d'un processus non branchant temporisé). *Soit  $\mathcal{A}$  un réseau d'automates temporisés et  $Dépliage$  le dépliage non temporisé de  $\mathcal{A}$ . Étant donné un événement  $t$  de  $Dépliage$ , nous définissons l'ensemble de variables d'horloges dépendant de  $t$ , noté  $V(t)$ , par  $V(t) = Var(\{t\} \cup \mathcal{C}^- \cup \mathcal{C}^+)$ .*

*Nous définissons la zone  $Z_t$  exprimant les contraintes locales liées à  $t$ , et définie sur l'ensemble d'horloges  $V(t)$ , comme la zone associée à la contrainte  $\varphi_t$  obtenue comme la conjonction des contraintes suivantes :*

*Contraintes causales :*

$$\begin{cases} \forall p \in t^\bullet, & \mathbf{d}_p(p) = \mathbf{d}_f(t) \\ \forall p \in \bullet t, & \mathbf{d}_p(p) \leq \mathbf{d}_f(t) \\ \forall p \in \bullet t \cap \mu^{-1}(X), & \mathbf{d}_c(p) = \mathbf{d}_f(t) \\ \forall p \in \circ t \cap \mu^{-1}(X), & \mathbf{d}_p(p) \leq \mathbf{d}_f(t) \leq \mathbf{d}_c(p) \end{cases}$$

*Contraintes temporisées <sup>8</sup> :*

$$\begin{cases} g(\mu(t))[\{x \leftarrow v_t^-(x)\}_{x \in X}] \\ \bigwedge_{\ell \in \ell^-} Inv(\ell)[\{x \leftarrow v_t^-(x)\}_{x \in X}] \\ \bigwedge_{x \in R(\mu(t))} v_t^+(x) = 0 \\ \bigwedge_{x \in Modif-Inv(\mu(t))} v_t^+(x) = v_t^-(x) \end{cases}$$

<sup>7</sup>Les zones ont été introduites et étudiées dans la sous-section 6.2.1 page 152.

<sup>8</sup>La notation  $g[x \leftarrow y]$  représente la substitution de l'horloge  $x$  par le terme  $y$  dans  $g$ .

Nous définissons par ailleurs la zone  $Z_0$ , sur l'ensemble de variables d'horloges  $V_0 = \text{Var}(\text{Min}(P))$ , comme la zone associée à la conjonction des contraintes suivantes :

$$\begin{cases} \forall p \in \text{Min}(P), & \mathbf{d}_p(p) = 0 \\ \forall p \in \text{Min}(P) \cap \mu^{-1}(X), & \mathbf{d}_r(p) = 0 \end{cases}$$

Étant donné un processus non branchant  $\beta$  de Dépliage, en remarquant que  $V(\beta) = V_0 \cup \bigcup_{t \in T} V(t)$ , nous définissons la zone  $W_\beta = Z_0 \cap (\bigcap_{t \in T} Z_t)$ . Comme précédemment, la bijection entre configurations et processus non branchants nous permet de définir la zone  $W_C$  associée à une configuration comme étant la zone  $W_\beta$  avec  $[\beta] = C$ .

Enfin, nous disons qu'un processus non branchant temporisé  $(\beta, v)$  est cohérent si et seulement si il vérifie  $v \in W_\beta$ .

**Remarque 7.23.** Considérons un événement  $t$  et la zone  $Z_t$  correspondante. Les variables d'horloges contraintes par  $Z_t$  sont uniquement des variables locales ( $V(t) = \text{Var}(\{t\} \cup C^- \cup C^+)$ ). Étant donnée une valuation  $v$  définie seulement sur cet ensemble de variables  $V(t)$ , l'appartenance de  $v$  à  $Z_t$  est stable sous décalage du temps global :

$$v \in Z_t \Rightarrow \forall \delta \in \mathbb{T}, v + \delta \in Z_t$$

┘

La proposition suivante établit le lien entre les séquences temporisées d'un réseau d'automates temporisés et ses processus non branchants temporisés cohérents. Elle constitue le résultat fondamental pour la correction de notre méthode.

**Proposition 7.24** (La cohérence temporelle est équivalente à l'exécution). *Soit  $\mathcal{A}$  un réseau d'automates temporisés et  $(\beta, v)$  un processus non branchant temporisé de  $\mathcal{A}$ . Nous avons alors l'équivalence suivante :*

$$(\beta, v) \text{ est cohérent} \iff \text{il existe une séquence temporisée } \sigma \text{ de } \mathcal{A} \text{ telle que } \text{Pr-T}(\sigma) = (\beta, v).$$

En d'autres termes, nous avons l'égalité suivante :

$$W_\beta = \{v \in \mathbb{T}^{V(\beta)} \mid \exists \sigma \text{ séquence temporisée de } \mathcal{A} \text{ telle que } \text{Pr-T}(\sigma) = (\beta, v)\}$$

**Preuve.** Supposons dans un premier temps qu'il existe une séquence temporisée  $\sigma$  de  $\mathcal{A}$  vérifiant  $\text{Pr-T}(\sigma) = (\beta, v)$ . Nous démontrons par induction sur la taille de  $\sigma$  que le processus non branchant temporisé  $\text{Pr-T}(\sigma)$  est cohérent. Ceci impliquera le résultat. Si  $\sigma$  est vide, alors le résultat est trivial. Sinon,  $\sigma$  s'écrit  $\sigma = \sigma'(\bar{t}, d)$  et par hypothèse d'induction le processus non branchant  $\text{Pr-T}(\sigma') = (\beta', v')$  est cohérent. Par définition de la construction inductive de  $\text{Pr-T}(\sigma)$  (définition 7.18), nous obtenons que  $\beta$  étend  $\beta'$  et que  $v$  coïncide (partiellement) avec  $v'$  sur les horloges de  $\beta'$ . Notons  $t$  l'événement de  $\beta$  étendant  $\beta'$ . Le seul point délicat pour démontrer la cohérence de  $(\beta, v)$  provient alors des invariants. En effet, dans les contraintes d'horloges définissant les zones  $Z_t$ , nous considérons les invariants associés à la coupe  $\mathcal{C} = \text{Cut}([t] \setminus \{t\})$ . Dans la définition de  $\text{Pr-T}(\sigma)$ , les invariants sont satisfaits pour la coupe  $\mathcal{C}' = \text{Cut}(\beta')$ . Ces deux coupes  $\mathcal{C}$  et  $\mathcal{C}'$  peuvent donc être différentes. Nous allons démontrer que les invariants correspondant sont cependant les mêmes. Les différences proviennent de franchissements d'événements  $t'$  n'appartenant pas à  $[t]$ , mais dont les dates de franchissement  $v(\mathbf{d}_F(t'))$  sont inférieures à la date  $v(\mathbf{d}_F(t))$ . De tels événements  $t'$  vérifient donc  $t' \text{co} < t$ . En particulier,  $t'$  ne peut ni consommer une place de  $\bullet t$  (sinon il serait en conflit avec  $t$ ),

ni consommer une place de  $\circ t$  (sinon il serait inférieur à  $t$  pour  $<$ ). En raisonnant par induction sur la taille de  $[\beta'] \setminus ([t] \setminus \{t\})$ , nous montrons donc que  $\beta'$  vérifie :

$$\bullet t \cup \circ t \subseteq \mathcal{C}'$$

Nous avons naturellement  $\bullet t \cup \circ t \subseteq \mathcal{C}$  d'après la définition de  $\mathcal{C}$  et donc d'après le lemme 7.14 nous obtenons finalement que les invariants correspondant sont les mêmes, ce qu'il fallait démontrer.

Réciproquement, supposons que le processus non branchant  $(\beta, v)$  soit cohérent. Nous construisons une séquence temporisée de  $\mathcal{A}$  ainsi :

1. associer  $v(\mathbf{d}_f(t))$  à l'occurrence de  $t$ ,
2. ordonner les transitions par rapport à l'ordre causal fort  $\prec$  de  $\beta$ ,
3. ordonner les transitions concurrentes dans  $\beta$  par rapport à leur date  $\mathbf{d}_f$ ,
4. ordonner arbitrairement les transitions concurrentes  $t$  et  $t'$  dans  $\beta$  de même date, *i.e.* telles que  $v(\mathbf{d}_f(t)) = v(\mathbf{d}_f(t'))$ .

Notons  $\sigma$  la séquence temporisée résultant de cette construction. Nous affirmons que  $\sigma$  est une séquence temporisée de  $\mathcal{A}$ . Il est facile de vérifier que cette séquence satisfait les contraintes imposées par les gardes et les remises à zéro. En revanche, le cas des invariants est à nouveau plus délicat, puisque les invariants exprimés dans la zone  $W_\beta$  (correspondant aux processus de  $[t]$  pour  $t$  événement de  $\beta$ ) ne correspondent pas nécessairement à ceux rencontrés dans  $\sigma$ . Cependant, le même raisonnement que celui développé pour le premier point permet, à l'aide du lemme 7.14, de démontrer que ces invariants sont en fait les mêmes.  $\square$

### 7.4.3 Définition et construction du dépliage temporisé

Nous pouvons à présent définir le dépliage temporisé d'un réseau d'automates temporisés  $\mathcal{A}$ . Celui-ci est obtenu à partir du dépliage non temporisé en attachant à chaque événement  $t$  du dépliage la zone  $W_{[t]}$  et supprimant l'ensemble des événements  $t$  tels que  $W_{[t]} = \emptyset$ . L'algorithme 6 calcule ce dépliage temporisé. Sa structure est très simple. Notons simplement que pour éviter d'explorer plusieurs fois une même extension qui se serait avérée incohérente du point de vue temporel, l'algorithme mémorise ces extensions en marquant l'événement associé « inutile ». Ceci implique que la procédure calculant les extensions possibles du dépliage n'explore pas les branches correspondantes.

Le lemme suivant découle d'une simple analyse des définitions.

**Lemme 7.25** (Calcul local des zones  $W_{[t]}$ ). *Étant donné un réseau d'automates temporisés  $\mathcal{A}$ , et un événement  $t$  du dépliage Dépliage de  $\mathcal{A}$ , définissons l'ensemble  $Max_{<t} = \{t' \in [t] \setminus \{t\} \mid \forall t'' \in [t] \setminus \{t\}, t' \not\prec t''\}$ . Nous avons alors*

$$W_{[t]} = Z_t \cap \bigcap_{t' \in Max_{<t}} W_{[t']}$$

Enfin, le théorème suivant énonce les propriétés vérifiées par le dépliage temporisé ainsi défini. Il découle facilement de la proposition 7.24.

**Théorème 7.26.** *Soit  $\mathcal{A}$  un réseau d'automates temporisés. L'algorithme 6 calcule le dépliage temporisé de  $\mathcal{A}$ , noté Dépliage-T( $\mathcal{A}$ ). Cet algorithme peut ne pas terminer. Le dépliage Dépliage-T( $\mathcal{A}$ ) vérifie les propriétés suivantes :*

- un événement  $t$  apparaît dans  $\text{Dépliage-T}(\mathcal{A})$  si et seulement si il existe une exécution temporisée dans  $\mathcal{A}$ , associée à une séquence temporisée  $\sigma$ , dont le processus temporisé  $\text{Pr-T}(\sigma) = (\beta, v)$  vérifie  $[\beta] = [t]$ ,
- la zone  $W_{[t]}$  associée à un événement  $t$  de  $\text{Dépliage-T}(\mathcal{A})$ , une fois projetée sur les variables  $\mathbf{d}_f(t')$ , pour  $t' \leq t$ , représente l'ensemble des dates possibles pour les séquences temporisées  $\sigma$  dont le processus non branchant est  $[t]$ .

---

**Algorithme 6** Construction du dépliage temporisé (semi-algorithme)
 

---

**Entrée :** Un réseau d'automates temporisés  $\mathcal{A}$ .

**Sortie :** Le dépliage temporisé  $\text{Dépliage-T}(\mathcal{A})$ .

```

1:  $\text{Dépliage-T} := \{(\ell_{1,0}, \emptyset), \dots, (\ell_{n,0}, \emptyset)\} \cup \{(x, \emptyset) \mid x \in X\};$            /* Initialisation */
2:  $ep := EP(\text{Dépliage-T});$                                                          /* Extensions possibles */
3: Tant Que  $ep \neq \emptyset$  Faire
4:   Choisir un événement  $t = (\bar{t}, Y_c, Y_l)$  dans  $ep$ .
5:   Calculer la zone  $W_{[t]}$ .                                                         /* À l'aide du lemme 7.25 */
6:   Si  $W_{[t]} \neq \emptyset$  Alors                                                   /* L'événement  $t$  est cohérent temporellement */
7:      $\text{Dépliage-T} := \text{Extension}(\text{Dépliage-T}, t);$                                /* Extension du processus */
8:      $ep := EP(\text{Dépliage-T});$                                                    /* Mise à jour de l'ensemble  $ep$  */
9:   Sinon                                                                           /* L'événement  $t$  n'est pas cohérent temporellement */
10:    Marquer  $t$  comme événement inutile.
11:   Fin Si
12: Fin Tant Que
13: Retourner  $\text{Dépliage-T};$ 

```

---

Nous décrivons enfin l'application des définitions précédentes aux exemples introduits dans la section 7.2.

**Exemple 7.27** (Exemple 7.3 poursuivi). Un préfixe du dépliage temporisé du réseau d'automates temporisés  $\mathcal{B}$  est représenté sur la figure 7.7. Seules les projections des zones sur les variables d'horloges associées aux événements sont représentées sur la figure. Considérons par exemple la zone associée à l'événement  $t_3$ . Nous obtenons comme attendu la caractérisation suivante :  $t_2$  peut survenir à n'importe quel instant positif ou nul.  $t_3$  intervient entre 0 et 1 unité de temps après  $t_2$ . Dans ce cas, le dépliage est infini et l'algorithme 6 ne termine donc pas. En effet, il est toujours possible d'étendre le dépliage avec un événement étiqueté par  $a$  en trouvant un  $\text{co}_{\prec}$ -ensemble formé d'une condition étiquetée par  $\ell'_0$  et d'une condition étiquetée par  $y$ . Notons de plus que la zone associée à ce dernier  $a$  est exprimée sur un nombre de variables égal au nombre d'occurrences de  $a$  précédentes. Ainsi, il est possible d'obtenir un nombre arbitrairement grand de variables.  $\lrcorner$

**Exemple 7.28** (Exemple 7.4 poursuivi). Le dépliage temporisé du réseau d'automates temporisés  $\mathcal{C}$  est représenté sur la figure 7.8. Celui-ci est fini et l'algorithme termine donc dans ce cas. De plus, comme nous l'avons souligné lors de la première analyse de ce réseau, la transition  $a$  ne peut avoir lieu qu'après la transition  $c_2$ . Nous l'obtenons ici grâce au système de contraintes associé à l'événement  $t_1$ . Celui-ci n'admet en effet aucune solution et la zone correspondante est donc vide. En particulier, cet événement ne sera donc pas construit par l'algorithme 6 (c'est pourquoi il est représenté en pointillés ici).  $\lrcorner$

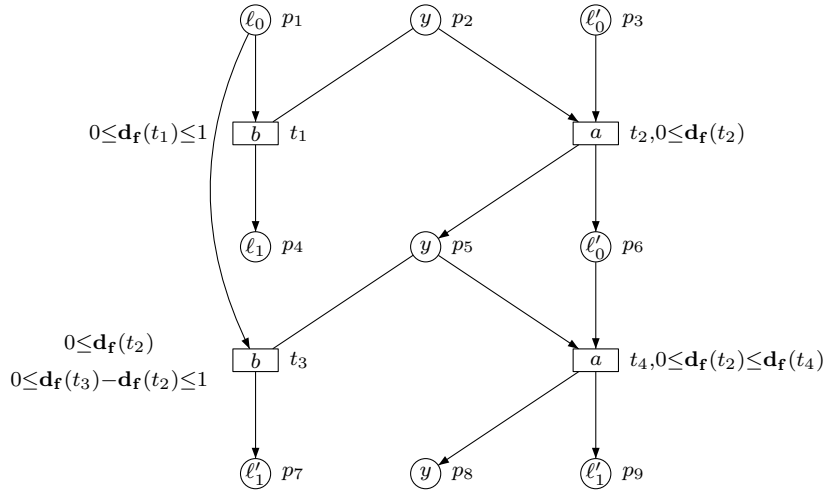


FIG. 7.7 – Un préfixe du dépliage temporisé du réseau  $\mathcal{B}$ .

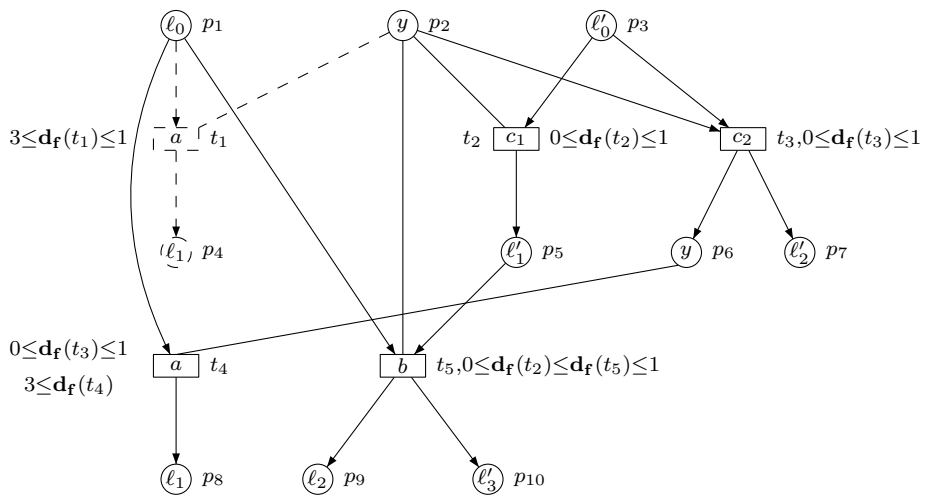


FIG. 7.8 – Le dépliage temporisé de  $\mathcal{C}$ .

#### 7.4.4 Construction du dépliage fondée sur des zones locales

La construction précédente souffre d'un défaut : les zones intervenant dans le dépliage ont un nombre de variables non borné. Or il est bien connu que les opérations élémentaires sur les zones telles que la normalisation ou le test du vide ont une complexité cubique dans le nombre de variables de la zone (voir la section 6.2 dans laquelle les opérations classiques sur les zones sont présentées). Ainsi, il est intéressant de proposer un algorithme ne manipulant que des zones définies sur un ensemble de variable de taille constante. Une idée naturelle consiste à restreindre les zones  $W_\beta$  à la coupe de  $\beta$ . De telles zones représentent alors l'état du système à l'issue de l'exécution de  $\beta$ , abstrayant l'ensemble des contraintes temporisées portant sur les événements du passé causal strict de  $\beta$ . Le principal intérêt de cette abstraction réside dans le nombre de variables d'horloges à considérer. Les variables associées à une coupe sont en effet en nombre constant égal à  $n + 3|X|$ , où  $n$  est le nombre d'automates du réseau et  $|X|$  est le cardinal de l'ensemble  $X$  des horloges du réseau. De plus, nous allons démontrer (lemme 7.31) que ces variables sont suffisantes pour étendre le processus non branchant  $\beta$ . Cependant, cette abstraction peut se révéler trop brutale lors du calcul de certaines zones. L'exemple 7.30 illustre les difficultés qui peuvent apparaître. Nous proposons dans le reste de cette partie un ensemble d'algorithmes permettant de calculer le dépliage temporisé en ne manipulant que des zones locales.

**Définition 7.29** (Zones locales). *Étant donné un réseau d'automates temporisés  $\mathcal{A}$  et un processus non branchant  $\beta$  du dépliage de  $\mathcal{A}$ , nous considérons l'ensemble des horloges associées à la coupe de  $\beta$ , noté  $\text{Loc-Var}(\beta) = \text{Var}(\text{Cut}(\beta))$ . La zone locale associée à  $\beta$ , notée  $W_\beta^{\text{loc}}$ , est alors définie comme la projection de la zone  $W_\beta$  sur les horloges  $\text{Loc-Var}(\beta)$ . Le nombre d'horloges impliquées dans la définition de la zone  $W_\beta^{\text{loc}}$  est égal à  $n + 3|X|$ , où  $n$  est le nombre d'automates du réseau et  $|X|$  est le nombre d'horloges de  $\mathcal{A}$ .*

Nous avons pu voir dans la définition des zones  $W_\beta$  correspondant aux processus non branchants  $\beta$  que ces zones s'obtiennent simplement comme les intersections des zones associées aux événements du processus. Nous illustrons les difficultés survenant avec les zones locales à l'aide de l'exemple 7.30.

**Exemple 7.30** (Difficultés liées aux zones locales). *Considérons le réseau d'automates temporisés  $\mathcal{A}$  représenté sur la sous-figure 7.9(a) ainsi que son dépliage représenté sur la sous-figure 7.9(b). Notons  $\beta_1$  (respectivement  $\beta_2$ ) le processus non branchant de configuration  $\{t_1, t_2, t_3, t_4\}$  (respectivement  $\{t_1, t_2, t_3, t_5\}$ ). Une simple analyse du comportement de  $\mathcal{A}$  permet d'obtenir les conclusions suivantes :*

- l'action  $b$  peut avoir lieu à toute date positive ou nulle,
- l'action  $c_1$  peut avoir lieu si et seulement si l'action  $b$  a lieu à la date 1,
- l'action  $c_2$  peut avoir lieu si et seulement si l'action  $b$  a lieu à la date 0,
- les deux actions  $c_1$  et  $c_2$  ne peuvent pas apparaître dans une même séquence temporisée.

Cette analyse montre qu'il n'existe aucun processus non branchant temporisé cohérent dont la configuration contient les événements  $t_4$  et  $t_5$ . Dans le dépliage de  $\mathcal{A}$ , les zones locales  $W_{\beta_1}^{\text{loc}}$  et  $W_{\beta_2}^{\text{loc}}$  associées respectivement aux processus non branchants  $\beta_1$  et  $\beta_2$  contraignent des ensembles de variables disjoints car les coupes des deux processus sont disjointes. En particulier, nous obtenons que l'intersection  $W_{\beta_1}^{\text{loc}} \cap W_{\beta_2}^{\text{loc}}$  de ces deux zones n'est pas vide. Elle ne représente donc pas les processus non branchants temporisés cohérents de configuration  $\{t_1, \dots, t_5\}$ . Ainsi, considérer simplement l'intersection des zones locales n'est pas suffisant. ┘

Le lemme suivant correspond au cas favorable pour l'extension d'un processus non branchant à partir de sa zone locale. Il énonce que pour un ensemble d'événements vérifiant les conditions

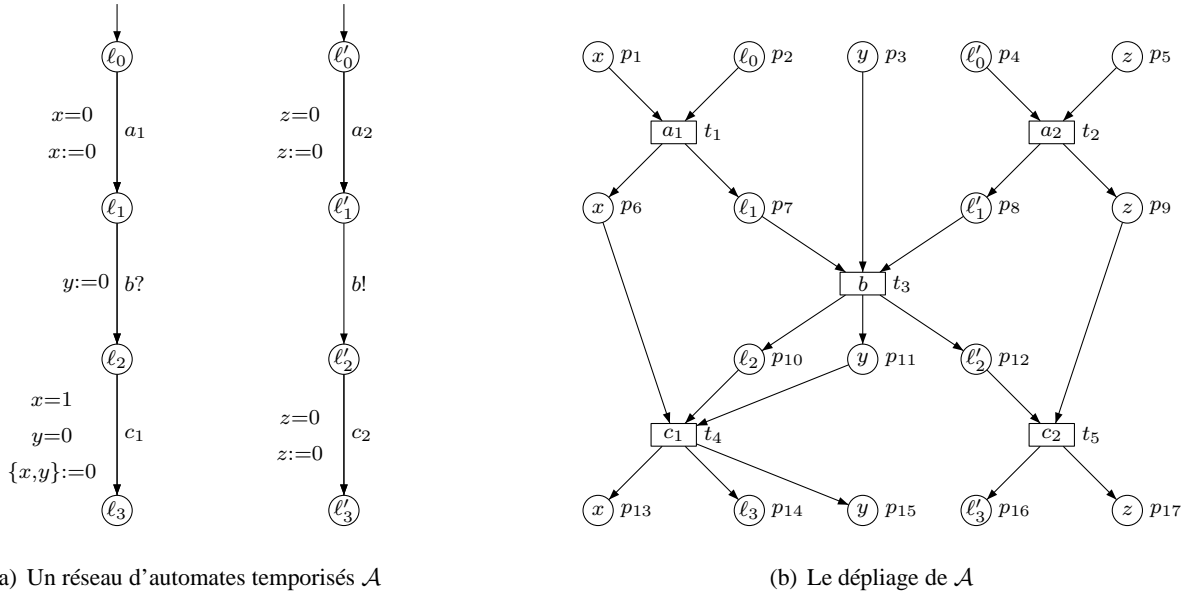


FIG. 7.9 – Difficultés liées aux zones locales.

(i) et (ii) ci-dessous, nous pouvons calculer en une seule étape la nouvelle zone locale obtenue à l'issue du franchissement de chacun des événements de cet ensemble. Ceci constitue donc un cas favorable pour le calcul des zones locales. Intuitivement, la condition (i) exprime le fait que tous les événements possèdent toutes leurs places en entrée dans la coupe de  $\beta$  et la condition (ii) exprime que ces événements sont faiblement concurrents, *i.e.* deux à deux incomparables pour la relation  $<$ . L'algorithme 7 présenté ensuite illustre ce lemme en explicitant le calcul de la zone locale obtenue à l'issue de ce franchissement.

**Lemme 7.31** (Extension par un ensemble d'événements faiblement concurrents). *Soit  $\mathcal{A}$  un réseau d'automates temporisés et  $\beta$  un processus non branchant du dépliage  $\text{Dépliage}(\mathcal{A})$ . Étant donné un ensemble  $T = \{t_1, \dots, t_k\}$  d'événements de  $\text{Dépliage}(\mathcal{A})$  vérifiant les conditions suivantes :*

- (i)  $\forall 1 \leq i \leq k, \bullet t_i \cup \circ t_i \subseteq \text{Cut}(\beta)$
- (ii)  $\forall 1 \leq i \neq j \leq k, t_i \text{co} < t_j$

*Nous disons alors que  $T$  constitue une tranche étendant  $\beta$ . De plus, en posant  $\beta_0 = \beta$ , nous avons pour tout indice  $i \in \{1, \dots, k\}$ ,  $t_i$  étend le processus non branchant  $\beta_{i-1}$  et nous posons  $\beta_i = \text{Extension}(\beta_{i-1}, t_i)$ . Nous affirmons enfin que la zone  $W_{\beta_k}^{\text{loc}}$  est la projection sur les horloges  $\text{Loc-Var}(\beta_k)$  de la zone  $W_{\beta}^{\text{loc}} \cap (\bigcap_{1 \leq i \leq k} Z_{t_i})$ .*

*Nous notons par la suite  $\text{Extension}(\beta, T)$  l'extension  $\beta_k$  du processus  $\beta$  par l'ensemble des événements de  $T$ .*

**Preuve.** Pour tout indice  $i$ , la proposition 3.2 de [Win02] entraîne que  $t_i$  étend le processus non branchant  $\beta_{i-1}$ . Le deuxième point est alors une simple conséquence des définitions 7.22 et 7.29.  $\square$

Le lemme suivant permet de décomposer un ensemble d'événements à ajouter en tranches maximales pour pouvoir appliquer le lemme 7.31.



**Algorithme 7** Extension par un ensemble d'événements faiblement concurrents – *Tir*

**Entrée :** Un quadruplet  $(\beta, \mathcal{C}, W, T)$  où  $\beta$  est un processus non branchant,  $\mathcal{C}$  sa coupe,  $W = W_{\beta}^{loc}$  sa zone locale et  $T$  est une tranche étendant  $\beta$ , selon les conditions du lemme 7.31.

**Sortie :** Le triplet  $(\beta', \mathcal{C}', W')$  avec  $\beta'$  l'extension de  $\beta$  par la tranche  $T$ ,  $\mathcal{C}'$  sa coupe et  $W' = W_{\beta'}^{loc}$  sa zone.

- 1:  $\beta' := \text{Extension}(\beta, T)$ ; /\*  $\beta'$  est l'extension de  $\beta$  par  $T$  \*/
- 2:  $\mathcal{C}' := (\mathcal{C} \setminus \bullet T) \cup T\bullet$ ; /\*  $\mathcal{C}'$  est la coupe de  $\beta'$  \*/
- 3: Soit  $\varphi$  une contrainte sur les horloges  $\text{Loc-Var}(\beta)$  représentant la zone  $W$ .
- 4: **Pour**  $t \in T$  **Faire**
- 5:     Définir la contrainte  $\varphi_t$  sur les horloges  $V(t)$  définissant la zone  $Z_t$ .
- 6: **Fin Pour**
- 7:  $\varphi' := \varphi \wedge (\bigwedge_{t \in T} \varphi_t)$ ;
- 8:  $W' := \llbracket \varphi' \rrbracket$ ;
- 9: Projeter la zone  $W'$  sur les variables d'horloges  $\text{Loc-Var}(\beta')$ .
- 10: **Retourner**  $(\beta', \mathcal{C}', W')$ .

**Lemme 7.32.** Soit  $\mathcal{A}$  un réseau d'automates temporisés et  $\beta$  et  $\beta'$  deux processus non branchants du dépliage  $\text{Dépliage}(\mathcal{A})$  tels que  $\beta \sqsubseteq \beta'$ . Notons  $T = [\beta'] \setminus [\beta]$  l'ensemble des événements présents dans  $\beta'$  mais absents dans  $\beta$ . Alors il existe une partition de  $T$  en  $m$  ensembles deux à deux disjoints  $\{T_1, \dots, T_m\}$  telle que pour tout indice  $i \in \{1, \dots, m\}$ ,  $T_i$  est une tranche maximale étendant le processus  $\beta_{i-1}$  en le processus  $\beta_i$  (en posant  $\beta_0 = \beta$ ) et vérifiant  $\beta_k = \beta'$ .

**Preuve.** Il suffit d'effectuer un tri topologique de l'ensemble  $E$  par rapport à la relation causale faible  $<$ . □

Rappelons que notre objectif global est de décrire une procédure permettant le calcul de la zone locale  $W_{[t]}^{loc}$  associée au processus non branchant minimal  $[t]$  de  $t$ . L'algorithme 8 présenté ici permet de réaliser ce calcul. Détaillons le fonctionnement de l'algorithme 8. Étant donné une extension possible  $t$ , notons  $\beta$  le processus non branchant correspondant à la configuration  $[t] \setminus \{t\}$ . Cet algorithme doit donc d'abord calculer la zone  $W_{\beta}^{loc}$  puis l'étendre en ajoutant l'événement  $t$ .

Les deux lemmes précédents nous permettent, étant donné deux processus  $\beta_1$  et  $\beta_2$  vérifiant  $\beta_1 \sqsubseteq \beta_2$ , de décomposer l'ensemble d'événements à ajouter afin d'appliquer pas à pas l'algorithme 7. Cependant, nous ne mémorisons dans le dépliage que les zones locales associées aux processus non branchants minimaux des événements, *i.e.* dont les configurations ont la forme  $[t']$  où  $t'$  est un événement. Nous devons donc trouver un événement  $t' \in [t] \setminus \{t\}$  tel que le processus  $\beta_{[t']}$  vérifie la propriété suivante :

$$\beta_{[t']} \sqsubseteq \beta \tag{7.1}$$

Nous proposons ici une procédure permettant de calculer un tel événement. D'après la définition 7.10, étant donné un événement  $t'$  de  $\beta$ , nous avons  $\beta_{[t']} \not\sqsubseteq \beta$  si et seulement s'il existe un événement  $t'' \in [\beta] \setminus [t']$  tel que  $t'' \prec t'$ . Ceci équivaut à l'existence d'une condition  $p \in \beta_{[t']} \setminus \text{Cut}(\beta_{[t']})$  lue par une transition  $t'' \in [\beta] \setminus [t']$ . Notre procédure calcule à l'aide de cette caractérisation, pour chaque événement maximal  $t_i$  de  $\beta$  pour  $<$ , un événement  $t'_i \leq t_i$  vérifiant la propriété (7.1). L'événement  $t'_i$  pour lequel l'ensemble  $[t'_i]$  est de taille maximale est alors sélectionné. Cette procédure est illustrée dans les lignes 1 à 10 de l'algorithme 8. Les lignes 11 à 19 appliquent ensuite les lemmes 7.32 et 7.31. La ligne 20 calcule simplement les successeurs par l'événement  $t$  du processus non branchant de configuration  $[t] \setminus \{t\}$ .

**Algorithme 8** Calculer la zone locale associée à une extension**Entrée :** Une extension possible  $t = (\bar{t}, Y_c, Y_l)$ .**Sortie :** La zone locale  $W_{[t]}^{loc}$  associée au processus non branchant  $[t]$ .

- 1: Calculer les événements  $t_1, \dots, t_m$  maximaux pour  $<$  dans l'ensemble  $[t] \setminus \{t\}$ .
- 2: **Pour**  $i \in \{1, \dots, m\}$  **Faire**
- 3:    $t' := t_i$ ;
- 4:   Soit  $\beta'$  le processus non branchant de  $[t']$ .
- 5:   **Tant Que**  $\exists p \in \beta' \setminus \text{Cut}(\beta'), \exists t'' \in [t] \setminus [t']$  tel que  $p \in {}^\circ t''$  **Faire**
- 6:      $t' := \bullet p$ ;
- 7:   **Fin Tant Que**
- 8:    $t'_i := t'$ ;
- 9: **Fin Pour**
- 10: Choisir  $k \in \{1, \dots, m\}$  pour lequel la taille de l'ensemble  $[t'_k]$  est maximale.
- 11: Soit  $\beta'$  le processus non branchant de  $[t'_k]$ .
- 12: Soit  $\mathcal{C}' = \text{Cut}([t'_k])$  et  $W' = W_{[t'_k]}^{loc}$ .
- 13: Calculer un tri topologique  $[T_1, \dots, T_{m'}]$  par rapport à  $<$  de l'ensemble  $[t] \setminus (\{t\} \cup [t'_k])$ .
- 14: **Pour**  $j \in \{1, \dots, m'\}$  **Faire**
- 15:    $(\beta', \mathcal{C}', W') := \text{Tir}(\beta', \mathcal{C}', W', T_j)$ ;
- 16:   **Si**  $W' = \emptyset$  **Alors**
- 17:     **Retourner**  $\emptyset$ ;
- 18:   **Fin Si**
- 19: **Fin Pour**
- 20:  $(\beta, \mathcal{C}, W) := \text{Tir}(\beta', \mathcal{C}', W', \{t\})$ ; /\*  $\beta$  vérifie  $[\beta] = [t]$  \*/
- 21: **Retourner**  $W$ ;

**Exemple 7.33** (Application de l'algorithme 8). Nous illustrons sur l'exemple représenté sur la figure 7.10 le fonctionnement des lignes 12 à 20 de l'algorithme 8 permettant de calculer un point de départ à l'application du calcul par tranche. Sur cet exemple, nous souhaitons ajouter l'événement  $t$ . Les trois événements maximaux sont les événements  $t_4, t_5$  et  $t_6$ . Le passé causal minimal  $[t_4]$  de  $t_4$  ne contient pas l'événement  $t_2$ . En effet, nous avons  $t_2 < t_4$  mais  $t_2 \not\prec t_4$ . Nous obtenons donc  $[t_4] = \{t_1, t_4\}$ . L'algorithme 8 obtient  $t'_4 = t_1$ . De même, nous obtenons  $t'_6 = t_6$ . Pour l'événement  $t_5$  enfin, nous avons  $[t_5] = \{t_1, t_2, t_3, t_5\}$  et l'algorithme calcule  $t'_5 = t_5$ . La ligne 11 sélectionne donc l'événement  $t'_5$ . L'ensemble d'événements à ajouter est  $T' = \{t_4, t_6\}$  qui constitue déjà une tranche. L'algorithme calcule donc les successeurs à l'issue de cette tranche, et obtient en un pas la zone locale associée au processus non branchant dont la configuration vaut  $\{t_1, \dots, t_6\}$ , ce qui était recherché afin de calculer la zone locale associée à l'événement  $t$ . ┘

Nous pouvons donc présenter notre nouvel algorithme de calcul du dépliage temporisé. Celui-ci, présenté comme l'algorithme 9, est une simple adaptation de l'algorithme 6. L'objet construit est différent : les zones attachées aux événements sont les zones locales et non les zones globales. Notons que la zone locale  $W_{[t]}^{loc}$  attachée à un événement  $t$  est vide si et seulement si la zone globale  $W_{[t]}$  l'est. Ce nouvel algorithme constitue donc une alternative à l'algorithme 6 présentant l'intérêt de n'impliquer que des zones définies sur un ensemble de variables de taille égale à  $n + 3|X|$  lors du test de la vacuité des zones<sup>9</sup>.

<sup>9</sup>En réalité, lors de l'application de l'algorithme 7, les zones intermédiaires peuvent faire intervenir un nombre plus grand de variables avant d'être projetées sur les variables de la coupe, mais qui demeure borné par  $2n + 5|X|$ .

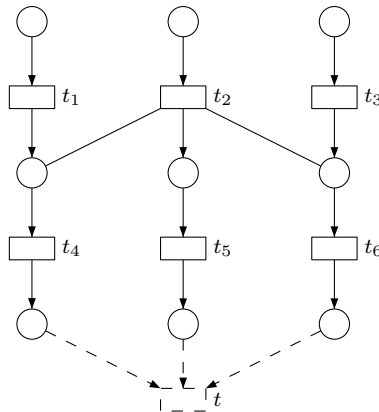


FIG. 7.10 – Un processus branchant

Nous obtenons enfin le théorème suivant, dont la preuve s'obtient facilement à l'aide des différents lemmes de correction précédents et à l'aide du théorème 7.26.

**Théorème 7.34.** *Soit  $\mathcal{A}$  un réseau d'automates temporisés. L'algorithme 9 calcule le dépliage temporisé avec zones locales de  $\mathcal{A}$ , noté  $\text{Dépliage-TL}(\mathcal{A})$ . Cet algorithme peut ne pas terminer. Le dépliage  $\text{Dépliage-TL}(\mathcal{A})$  vérifie les propriétés suivantes :*

- un événement  $t$  apparaît dans  $\text{Dépliage-TL}(\mathcal{A})$  si et seulement s'il existe une exécution temporisée dans  $\mathcal{A}$ , associée à une séquence temporisée  $\sigma$ , dont le processus temporisé  $\text{Pr-T}(\sigma) = (\beta, v)$  vérifie  $[\beta] = [t]$ ,
- la zone  $W_{[t]}^{\text{loc}}$  associée à un événement  $t$  de  $\text{Dépliage-TL}(\mathcal{A})$  représente l'ensemble des valeurs possibles pour les horloges associées à la coupe de  $[t]$  obtenues par les séquences temporisées  $\sigma$  dont le processus non branchant temporisé vérifie  $\text{Pr-T}(\sigma) = ([t], v)$ .

---

**Algorithme 9** Construction du dépliage temporisé avec zones locales (semi-algorithme)
 

---

**Entrée :** Un réseau d'automates temporisés  $\mathcal{A}$ .

**Sortie :** Le dépliage temporisé « local »  $\text{Dépliage-TL}(\mathcal{A})$ .

- 1:  $\text{Dépliage-TL} := \{(\ell_{1,0}, \emptyset), \dots, (\ell_{n,0}, \emptyset)\} \cup \{(x, \emptyset) \mid x \in X\}$ ; /\* Initialisation \*/
  - 2:  $ep := EP(\text{Dépliage-TL})$ ; /\* Extensions possibles \*/
  - 3: **Tant Que**  $ep \neq \emptyset$  **Faire**
  - 4:   Choisir un événement  $t = (\bar{t}, Y_c, Y_l)$  dans  $ep$ .
  - 5:   Calculer la zone  $W_{[t]}^{\text{loc}}$  à l'aide de l'algorithme 8.
  - 6:   **Si**  $W_{[t]}^{\text{loc}} \neq \emptyset$  **Alors** /\*  $t$  temporellement cohérent \*/
  - 7:      $\text{Dépliage-TL} := \text{Extension}(\text{Dépliage-TL}, t)$ ;
  - 8:      $ep := EP(\text{Dépliage-TL})$ ; /\* Mise à jour de l'ensemble  $ep$  \*/
  - 9:   **Sinon** /\*  $t$  temporellement incohérent \*/
  - 10:   Marquer  $t$  comme événement inutile.
  - 11:   **Fin Si**
  - 12: **Fin Tant Que**
  - 13: Retourner  $\text{Dépliage-TL}$ ;
-

## 7.5 Préfixe fini et complet

Afin de pouvoir utiliser le dépliage temporisé décrit précédemment, il est nécessaire de pouvoir se limiter à un préfixe fini de sorte à obtenir des algorithmes qui terminent. Nous nous intéressons donc maintenant à la construction d'un préfixe fini du dépliage temporisé qui soit complet, *i.e.* représentant l'ensemble des configurations accessibles dans  $\mathcal{A}$ .

### 7.5.1 Introduction

**Définitions.** Nous commençons par définir les notions de préfixe fini et complet. Dans la suite,  $\mathcal{A}$  représente un réseau d'automates temporisés.

**Définition 7.35** (Préfixe fini). *Soit  $\mathcal{N}$  un sous-ensemble du dépliage  $\text{Dépliage}(\mathcal{A})$ , alors nous disons que  $\mathcal{N}$  est un préfixe de  $\text{Dépliage}(\mathcal{A})$  si et seulement si il vérifie les conditions suivantes :*

$$\begin{cases} \text{Min}(\text{Dépliage}(\mathcal{A})) \subseteq \mathcal{N} \\ \bullet\mathcal{N} \cup \circ\mathcal{N} \subseteq \mathcal{N} \end{cases}$$

*Si de plus  $\mathcal{N}$  est fini, alors nous disons que  $\mathcal{N}$  constitue un préfixe fini de  $\text{Dépliage}(\mathcal{A})$ .*

**Définition 7.36** (Préfixe complet). *Soit  $\mathcal{N}$  un préfixe du dépliage  $\text{Dépliage}(\mathcal{A})$ , alors nous disons que  $\mathcal{N}$  est un préfixe complet de  $\text{Dépliage}(\mathcal{A})$  si et seulement si il vérifie la condition suivante<sup>10</sup> :*

$$\forall (\ell, \nu) \in \text{Acc}(\mathcal{A}), \exists (\beta, \nu) \text{ cohérent, tel que } [\beta] \in \mathcal{N} \text{ et } \text{Config}(\beta, \nu) = (\ell, \nu)$$

où  $\text{Acc}(\mathcal{A})$  représente l'ensemble des configurations accessibles depuis la configuration initiale dans le réseau d'automates temporisés  $\mathcal{A}$  (défini page 50).

Avec ces définitions, les théorèmes 7.26 et 7.34 entraînent le résultat suivant.

**Corollaire 7.37** (Complétude). *Les dépliages  $\text{Dépliage-T}$  et  $\text{Dépliage-TL}$  obtenus respectivement par les algorithmes 6 et 9 sont complets mais pas nécessairement finis.*

**Événements redondants.** Dans le cadre non temporisé, divers travaux ont étudié le problème de la caractérisation d'un préfixe fini et complet. En effet, celui-ci est essentiel pour toutes les perspectives de vérification du système étudié. McMillan proposa dans [McM95] une première notion d'événement « cut-off », *redondant*, définie pour un événement  $t$  par :

$$t \text{ est redondant si } \exists t' \text{ tel que } \begin{cases} \mu(\text{Cut}([t])) = \mu(\text{Cut}([t'])) \\ |[t']| < |[t]| \end{cases}$$

Intuitivement, un événement est redondant au sens où il existe un autre événement du dépliage le représentant déjà. Ceci permet de stopper la construction du dépliage au delà de certains événements et ainsi d'obtenir un préfixe fini du dépliage qui contient une occurrence de chaque événement tirable. Plusieurs travaux ont alors permis d'obtenir des préfixes plus petits, en proposant des définitions légèrement différentes pour ces événements redondants, fondées sur la notion d'ordre adéquat [ERV02]. Ces ordres permettent de comparer les événements et se substituent au deuxième point de la définition de la redondance. Un des corollaires importants de ces nouveaux préfixes est la propriété suivante

<sup>10</sup>Dans cette définition, la cohérence est définie par rapport au dépliage temporisé  $\text{Dépliage-T}(\mathcal{A})$ .

portant sur la taille du préfixe : le nombre d'événements apparaissant dans le préfixe est inférieur au nombre de marquages accessibles du réseau de Petri sauf déplié [ERV02, Proposition 5.3].

En présence d'arcs de lecture, les travaux de [VSY98] ont montré que les constructions précédentes ne s'appliquent pas à l'ensemble de la classe des réseaux de Petri saufs avec arcs de lecture, mais seulement à une sous-classe. Plus récemment, Winkowski a démontré dans [Win02] qu'il était possible d'obtenir des préfixes finis et complets pour l'ensemble de la classe des réseaux de Petri saufs avec arcs de lecture. Nous allons dans la suite utiliser cette méthodologie afin d'obtenir une construction pour les dépliages de réseaux d'automates temporisés. Cependant, la construction proposée dans [Win02] est plus complexe et sa complexité est plus élevée. En effet, les arcs de lecture peuvent être ajoutés « dans le passé » et il est donc nécessaire de s'assurer que même après ces ajouts, les événements redondants le sont encore.

**Adaptation au cadre temporisé.** Afin de prendre en compte les contraintes temporelles dans la comparaison des événements, il semble naturel d'utiliser les zones associées aux processus non branchants et d'utiliser un test d'inclusion comme cela est fait dans l'algorithme standard d'analyse en avant des automates temporisés (voir algorithme 2 page 155). Cependant, les contraintes représentées par les zones  $W_\beta$  ou  $W_\beta^{loc}$  obtenues précédemment ne sont pas comparables puisqu'elles portent sur des dates absolues. Nous allons présenter dans la section suivante la notion de zone relativisée permettant de résoudre ce problème. Comme le nombre de ces zones est a priori infini, nous allons également devoir, comme pour l'algorithme 2 d'analyse en avant à la volée, utiliser l'opérateur d'extrapolation. Nous verrons que l'application de cet opérateur pose des problèmes dans le cadre des dépliages. Enfin, notons une autre difficulté due à l'aspect temporisé. Il est possible d'étendre un processus non branchant à l'aide d'un événement dont l'occurrence peut se situer dans le passé temporel du processus non branchant. Par exemple, si l'événement ajouté  $t'$  est (faiblement ou fortement) concurrent à l'événement  $t$  qui est étendu, la date de franchissement de  $t'$  peut être inférieure à celle de  $t$ . Ceci justifie que dans la suite, nous devons comparer non seulement les valeurs des horloges obtenues à l'issue d'un processus non branchant mais aussi les dates de production des places de l'ensemble de la coupe finale.

Dans un premier temps, nous montrons comment l'algorithme proposé dans [Win02] permet d'obtenir une solution pour la classe des automates temporisés bornés. Dans un second temps, nous présentons une solution originale fondée sur la notion d'événements synchronisés qui permet de s'affranchir du passé de l'événement.

### 7.5.2 Automates bornés

La méthode de [Win02] ne permet pas dans le cas général d'obtenir un préfixe fini car le nombre de zones mises en jeu n'est pas fini. Cependant, pour la sous-classe des réseaux d'automates temporisés bornés, ce nombre est fini et nous allons montrer que cela permet d'obtenir un préfixe fini et complet.

**Définition 7.38** (Réseau d'automates temporisés bornés). *Étant donné un réseau d'automates temporisés  $\mathcal{A}$ , celui-ci est dit borné s'il existe une constante  $K$  telle que le long de toute exécution temporisée, toutes ses horloges et tous les temps de séjour dans un état de contrôle sont bornés par  $K$ .*

Notons qu'il est facile d'obtenir des conditions syntaxiques assurant qu'un réseau d'automates temporisés est borné. Cette sous-classe constitue une restriction classique, qui est expressivement équivalente du point de vue des langages à la classe générales des réseaux d'automates temporisés. Il est toujours possible, étant donné un automate temporisé  $\mathcal{A}$ , de construire un automate temporisé

borné  $\mathcal{A}'$  tel que  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ . Cependant, cette construction nécessite d'ajouter des transitions silencieuses et des invariants au système et peut donc dégrader certaines propriétés du modèle initial. Ceci est en particulier le cas pour les propriétés d'indépendance entre les différents automates d'un réseau d'automates temporisés. Dans la suite de cette sous-section, les réseaux considérés seront des réseaux d'automates temporisés bornés.

Afin de comparer deux événements ayant potentiellement lieu à des instants indépendants, il est nécessaire de relativiser les zones qui leur sont associées par rapport à la date de franchissement de l'événement correspondant. Plus généralement, nous aurons besoin de comparer des processus non branchants ne correspondant pas nécessairement à des processus non branchants minimaux. Nous adaptons donc la notion de processus non branchant primitif [Win02] qui permet de sélectionner les processus non branchants ayant un unique événement « final » au cadre temporisé.

**Définition 7.39** (Éléments primitifs). *Soit  $\mathcal{A}$  un réseau d'automates temporisés,  $\text{Dépliage}(\mathcal{A})$  le dépliage de  $\mathcal{A}$  et  $C$  une configuration de  $\mathcal{P}$ . Cette configuration est dite primitive si elle admet un unique événement maximal  $t$  vis-à-vis de la restriction à  $C$  de la relation causale forte  $\preceq$  de  $\text{Dépliage}(\mathcal{A})$ .*

*L'ensemble des configurations primitives d'événement maximal  $t$  est noté  $\text{Prim}(t)$ .*

*Enfin, nous dirons qu'un processus non branchant est primitif si sa configuration l'est.*

**Remarque 7.40.** Pour une configuration primitive  $C$  d'événement maximal  $t$ , tout processus non branchant temporisé cohérent  $(\beta, v)$  tel que  $[\beta] = C$  vérifie la propriété suivante :

$$\forall \mathbf{d} \in \text{Var}(\beta), \mathbf{d} \neq \mathbf{d}_c(p) \text{ avec } p \in \text{Cut}(\beta) \Rightarrow v(\mathbf{d}_f(t)) \geq v(\mathbf{d})$$

┘

**Lemme 7.41.** *Soit  $C$  une configuration et  $t \in C$  un événement. Alors il existe une configuration  $C' \in \text{Prim}(t)$  vérifiant  $C' \sqsubseteq C$ .*

**Preuve.** Il suffit de considérer  $C' = \{t' \in C \mid t' \preceq t\}$ . □

Nous définissons à présent la zone relativisée associée à une configuration primitive  $C$  d'événement maximal  $t$ . Intuitivement, celle-ci contient les contraintes temporelles associées à la coupe de  $C$  relativisées par rapport à la date absolue à laquelle la transition  $t$  a lieu. De plus, les dates de consommation des places sont éliminées. En effet, celles-ci ne sont pas contraintes lorsque seule la configuration  $C$  est étudiée.

**Définition 7.42** (Zone relativisée). *Soit  $C$  une configuration primitive d'événement maximal  $t$ . Nous définissons l'ensemble d'horloges  $V(C) = \{\mathbf{d} \in \text{Loc-Var}(\beta_C) \mid \mathbf{d} \neq \mathbf{d}_c(p) \text{ pour } p \in \text{Cut}(C)\}$ . Nous désignons par  $\varepsilon(\mathbf{d})$  une copie de l'horloge  $\mathbf{d}$ . Nous définissons la zone relativisée associée à  $C$ , notée  $\text{Relativisée}(C)$ , comme la zone associée à la contrainte d'horloge suivante :*

$$\exists \text{Var}(\beta_C) . \varphi_{\beta_C} \wedge \left( \bigwedge_{\mathbf{d} \in V(C)} \varepsilon(\mathbf{d}) = \mathbf{d}_f(t) - \mathbf{d} \right)$$

où  $\varphi_{\beta_C}$  est une contrainte d'horloge représentant la zone  $W_{\beta_C}$ . Enfin,  $\text{Relativisée}(C)$  est une zone sur l'ensemble d'horloges  $V(C)$ .

**Remarque 7.43** (Zone relativisée et zones locales). Il est possible de calculer la zone relativisée à partir de la zone locale correspondante. En effet, étant donnée une configuration primitive d'événement maximal  $t$ , il existe une condition  $p \in t^\bullet$ . Fixons une telle condition. La contrainte d'horloges suivante définit alors la zone *Relativisée*( $C$ ) :

$$\exists \text{Loc-Var}(\beta_C) \cdot \varphi_{\beta_C}^{\text{loc}} \wedge \left( \bigwedge_{\mathbf{d} \in V(C)} \varepsilon(\mathbf{d}) = \mathbf{d}_p(p) - \mathbf{d} \right)$$

où  $\varphi_{\beta_C}^{\text{loc}}$  est une contrainte d'horloge représentant la zone  $W_{\beta_C}^{\text{loc}}$ . Cette propriété, indépendante du choix de la condition  $p$ , est obtenue grâce à l'égalité de la date de production de la condition  $p$  ( $\mathbf{d}_p(p)$ ) et de la date de franchissement de  $t$  ( $\mathbf{d}_f(t)$ ).  $\dashv$

Le lemme suivant établit le résultat attendu pour les automates bornés : les zones relativisées sont en nombre fini. La preuve de ce résultat s'obtient en constatant que les horloges présentes dans les zones relativisées correspondent ou bien aux horloges du réseau ou bien aux temps de séjour dans les états de contrôle et que ces deux quantités sont bornées par définition.

**Lemme 7.44.** *Soit  $\mathcal{A}$  un réseau d'automates temporisés bornés et  $\text{Dépliage}(\mathcal{A})$  son dépliage (non temporisé). Alors l'ensemble des zones *Relativisée*( $C$ ),  $C$  décrivant l'ensemble des configurations primitives de  $\text{Dépliage}(\mathcal{A})$ , est fini.*

Nous pouvons à présent adapter la notion de configuration redondante introduite dans [Win02] à notre cadre.

**Définition 7.45** (Configuration redondante). *Soit  $\mathcal{A}$  un réseau d'automates temporisés,  $\text{Dépliage}(\mathcal{A})$  le dépliage de  $\mathcal{A}$  et  $C$  une configuration primitive de  $\text{Dépliage}(\mathcal{A})$ . Nous dirons que cette configuration est redondante s'il existe une sous-configuration primitive (stricte)  $C'$  vérifiant les trois conditions suivantes :*

$$\begin{cases} C' \sqsubseteq C \\ \mu(\text{Cut}(C')) = \mu(\text{Cut}(C)) \\ \text{Relativisée}(C) \subseteq \text{Relativisée}(C') \end{cases} \quad (7.2)$$

Intuitivement, cela signifie que d'une part  $C$  est construite après  $C'$  et que d'autre part l'ensemble des configurations du réseau  $\mathcal{A}$  représentées par le couple  $(\text{Cut}(C), \text{Relativisée}(C))$  est inclus dans l'ensemble des configurations du réseau  $\mathcal{A}$  représentées par le couple  $(\text{Cut}(C'), \text{Relativisée}(C'))$ . Ainsi, la seconde configuration  $C$  est « inutile » puisqu'elle présente de la redondance.

Le résultat fondamental est le suivant. Il énonce que l'information mémorisée dans les zones relativisées est suffisante pour calculer les extensions d'un processus non branchant primitif.

**Lemme 7.46** (Extension). *Soient  $C$  et  $C'$  deux configurations primitives vérifiant les conditions de redondances (7.2). Alors pour tout processus non branchant temporisé cohérent  $(\beta, v)$  tel que  $C \sqsubseteq [\beta]$  (noté  $[\beta] = C \oplus Y$ ), il existe un processus non branchant temporisé cohérent  $(\beta', v')$  vérifiant :*

$$\begin{cases} [\beta'] = C' \oplus Y' \\ Y' \text{ et } Y \text{ isomorphes via } \mu \\ \text{Config}(\beta', v') = \text{Config}(\beta, v) \end{cases} \quad (7.3)$$

**Preuve.** Notons  $(\beta_1, v_1)$  la restriction de  $(\beta, v)$  à la configuration  $C$ . Puisque  $(\beta, v)$  est cohérent,  $(\beta_1, v_1)$  l'est également et nous obtenons  $v_1 \in W_{\beta_1}$ . Notons  $t$  (respectivement  $t'$ ) représente l'événement maximal pour  $\prec$  de  $C$  (respectivement de  $C'$ ). Nous introduisons les notations suivantes :

- $V = V(C)$ ,
- $V' = V(C')$ ,
- $\eta$  dénote la bijection canonique entre  $V$  et  $V'$ .

L'existence de l'application  $\eta$  est assurée par la propriété  $\mu(\text{Cut}(C)) = \mu(\text{Cut}(C'))$ . L'inclusion  $\text{Relativisée}(C) \subseteq \text{Relativisée}(C')$  implique alors l'existence d'un processus non branchant temporisé cohérent  $(\beta'_1, v'_1)$  vérifiant les propriétés suivantes, en posant  $\delta = v_1(\mathbf{d}_f(t)) - v'_1(\mathbf{d}_f(t'))$  :

$$\begin{cases} [\beta'_1] = C' \\ v'_{1|V'} \circ \eta + \delta = v_{1|V} \end{cases}$$

où la notation  $v|_V$  représente la restriction de la valuation  $v$  au sous-ensemble de variables  $V$ . D'après la remarque 7.23 énonçant que l'extension d'un processus non branchant temporisé cohérent est stable par décalage du temps global, nous pouvons démontrer par induction sur la taille de  $Y$  qu'il est possible d'étendre le processus  $(\beta'_1, v'_1)$  en un processus non branchant temporisé cohérent  $(\beta', v')$  vérifiant les propriétés annoncées, ce qui conclut la preuve de ce lemme.  $\square$

**Définition 7.47.** Soit  $t$  un événement du dépliage  $\text{Dépliage}(\mathcal{A})$ . Alors,  $t$  est dit

- redondant si chacune des configurations primitives appartenant à  $\text{Prim}(t)$  est redondante,
- informatif s'il existe une configuration primitive appartenant à  $\text{Prim}(t)$  qui n'est pas redondante.

Soulignons qu'un événement est informatif si et seulement s'il n'est pas redondant.

**Définition 7.48** (Préfixe du dépliage). Soit  $\mathcal{A}$  un réseau d'automates temporisés et  $\text{Dépliage}(\mathcal{A})$  son dépliage. Notons  $\text{Info}$  l'ensemble des événements informatifs de  $\text{Dépliage}(\mathcal{A})$ . Nous définissons le préfixe suivant de  $\text{Dépliage}(\mathcal{A})$  :

$$\text{Préfixe-B}(\mathcal{A}) = \text{Min}(\text{Dépliage}(\mathcal{A})) \cup \text{Info} \cup \bullet\text{Info} \cup \circ\text{Info} \cup \text{Info}\bullet$$

**Théorème 7.49** (Finitude). Le préfixe  $\text{Préfixe-B}(\mathcal{A})$  est fini.

**Preuve.** Raisonnons par l'absurde et supposons que l'ensemble des événements informatifs  $\text{Info}$  soit infini. Alors l'ensemble des configurations primitives qui ne sont pas redondantes est également infini. Puisque chaque configuration de cet ensemble possède un nombre fini de successeurs directs dans ce même ensemble, il existe une chaîne infinie

$$C_1 \sqsubseteq C_2 \sqsubseteq \dots$$

telle que pour tout indice  $i$ ,  $C_i$  n'est pas redondante et est différente de  $C_{i+1}$ . D'après le lemme 7.44, l'ensemble des zones relativisées est fini. De plus, l'ensemble des images par  $\mu$  des coupes est également fini puisque  $\mu$  est à valeurs dans un ensemble fini. Ainsi, il existe nécessairement deux indices  $i < j$  pour lesquels les configurations  $C_i$  et  $C_j$  vérifient les conditions suivantes, qui impliquent les conditions de redondances (7.2) :

$$\begin{cases} C_i \sqsubseteq C_j \\ \mu(\text{Cut}(C_i)) = \mu(\text{Cut}(C_j)) \\ \text{Relativisée}(C_j) = \text{Relativisée}(C_i) \end{cases}$$

Ceci implique que  $C_j$  est une configuration redondante, ce qui fournit une contradiction. Ainsi, l'ensemble des événements informatifs  $\text{Info}$  est fini. Finalement, le préfixe  $\text{Préfixe-B}(\mathcal{A})$  est également fini.  $\square$



**Théorème 7.50** (Complet). *Le préfixe  $\text{Préfixe-B}(\mathcal{A})$  est complet.*

**Preuve.** Le dépliage  $\text{Dépliage}(\mathcal{A})$  est complet d'après le corollaire 7.37. Considérons donc une configuration  $(\ell, \nu)$  du réseau  $\mathcal{A}$ . Celle-ci est donc présente dans  $\text{Dépliage}(\mathcal{A})$ , *i.e.* il existe un processus non branchant temporisé cohérent  $(\beta, v)$  tel que la configuration  $[\beta]$  appartienne à  $\text{Dépliage}(\mathcal{A})$  et tel que  $\text{Config}(\beta, v) = (\ell, \nu)$ . Notons  $C = [\beta]$ . Deux cas peuvent se produire. Si  $C$  est une configuration ne contenant pas d'événement redondant, alors  $C$  appartient à  $\text{Préfixe-B}(\mathcal{A})$  et nous obtenons le résultat. Sinon,  $C$  contient un événement redondant  $t$ . D'après le lemme 7.41, il existe une configuration  $C' \in \text{Prim}(t)$  vérifiant  $C' \sqsubseteq C$ . Nous écrivons  $C = C' \oplus Y$ . De plus comme  $t$  est redondant,  $C'$  l'est, et il existe donc une sous-configuration primitive (stricte)  $C''$  de  $C'$  vérifiant les conditions (7.2) :

$$\begin{cases} C'' \sqsubseteq C' \\ \mu(\text{Cut}(C'')) = \mu(\text{Cut}(C')) \\ \text{Relativisée}(C') \subseteq \text{Relativisée}(C'') \end{cases}$$

Le lemme 7.46 s'applique alors à  $(\beta, v)$ . Nous obtenons ainsi un processus non branchant temporisé cohérent  $(\beta', v')$  vérifiant les conditions (7.3) :

$$\begin{cases} [\beta'] = C'' \oplus Y' \\ Y' \text{ et } Y \text{ isomorphes via } \mu \\ \text{Config}(\beta', v') = \text{Config}(\beta, v) = (\ell, \nu) \end{cases}$$

Ainsi, la configuration  $[\beta']$  contient strictement moins d'événements que la configuration  $C$  et permet également d'atteindre la configuration  $(\ell, \nu)$ . En répétant cette construction à partir de  $(\beta', v')$ , nous obtenons comme souhaité un processus non branchant temporisé cohérent dont la configuration appartient à  $\text{Préfixe-B}(\mathcal{A})$  puisque la taille de la configuration  $[\beta]$  ne peut pas diminuer infiniment.  $\square$

**Présentation de l'algorithme.** Nous pouvons maintenant décrire l'algorithme général obtenu pour le calcul du préfixe fini et complet  $\text{Préfixe-B}(\mathcal{A})$  du dépliage temporisé  $\text{Dépliage}(\mathcal{A})$  de  $\mathcal{A}$ . Celui-ci est donné par l'algorithme 10. Il est présenté ici à partir de la construction fondée sur les zones globales mais s'adapte naturellement au cas des zones locales  $W_\beta^{\text{loc}}$ . Notons que nous devons mémoriser pour chaque événement s'il est informatif ou redondant. De plus, lors du calcul des extensions possibles, seuls les événements informatifs sont pris en compte. Enfin, une opération supplémentaire apparaît ligne 12 consistant à mettre à jour le statut des événements du préfixe. En effet, un événement redondant  $t$  peut devenir informatif après l'ajout d'un événement dans le préfixe si cet ajout crée de nouvelles configurations primitives appartenant à  $\text{Prim}(t)$ . Cependant, le contraire n'est pas vrai, *i.e.* un événement informatif ne peut pas devenir redondant, car étant donnée une configuration, l'ensemble des configurations plus petites (pour  $\sqsubseteq$ ) ne peut pas être modifié. Cet algorithme termine donc puisque l'ensemble des événements informatifs est fini.

### 7.5.3 Événements synchronisés

Comme nous l'avons vu précédemment, la méthode fondée sur les automates bornés ne s'applique pas directement aux réseaux d'automates temporisés généraux. Nous décrivons donc à présent une deuxième approche sensiblement différente de la précédente. Afin d'éviter la difficulté introduite par le temps qui permet d'ajouter des événements dans le passé temporel d'un événement donné, nous considérons cette fois un sous-ensemble d'événements, appelés événements synchronisés, pour lesquels ce problème ne peut pas survenir. En effet, lorsqu'un événement  $t$  synchronise l'ensemble des

---

**Algorithme 10** Construction d'un préfixe fini et complet du dépliage temporisé pour les automates bornés

---

**Entrée :** Un réseau d'automates temporisés bornés  $\mathcal{A}$ .

**Sortie :** Le préfixe fini et complet  $Préfixe-B(\mathcal{A})$  du dépliage temporisé de  $\mathcal{A}$ .

```

1:  $Préfixe-B := \{(\ell_{1,0}, \emptyset), \dots, (\ell_{n,0}, \emptyset)\} \cup \{(x, \emptyset) \mid x \in X\};$  /* Initialisation */
2:  $ep := EP(Préfixe-B);$  /* Extensions possibles */
3: Tant Que  $ep \neq \emptyset$  Faire
4:   Choisir un événement  $t = (\bar{t}, Y_c, Y_l)$  dans  $ep$ .
5:   Si  $W_{[\bar{t}]} \neq \emptyset$  Alors /*  $t$  est temporellement cohérent */
6:      $Préfixe-B := Extension(Préfixe-B, t);$ 
7:     Si  $\forall C \in Prim(t)$ ,  $C$  est redondante Alors /*  $t$  est redondant */
8:       Marquer  $t$  comme redondant.
9:     Sinon /*  $t$  est informatif */
10:      Marquer  $t$  comme informatif
11:     Fin Si
12:     Mettre à jour les états « redondant », « informatif » des événements.
13:      $ep := EP(Préfixe-B);$  /* Mise à jour de l'ensemble  $ep$  */
14:     Sinon /*  $t$  est temporellement incohérent */
15:       Marquer  $t$  comme événement inutile.
16:     Fin Si
17: Fin Tant Que
18: Retourner  $Préfixe-B$ ;

```

---

processus du réseau, les dates de franchissement de tous les événements  $t'$  étendant  $t$  sont plus grandes que la date de franchissement de  $t$ . Cette propriété est l'ingrédient clé qui nous permet par la suite d'obtenir un préfixe fini. Remarquons que cette observation est proche de celle réalisée dans [LNZ04] concernant l'opérateur  $\$$ .

Nous définissons un ensemble de transitions « inévitables ».

**Définition 7.51.** Soit  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$  un réseau d'automates temporisés et  $S'$  un sous-ensemble de l'ensemble des transitions synchronisées de  $\mathcal{A}$ , i.e.  $S' \subseteq Sync$ , alors  $S'$  est dit inévitable si et seulement si pour tout indice  $i$ , tout circuit du graphe sous-jacent à  $\mathcal{A}_i$  intersecte  $S'$  : il existe une transition  $t_i$  appartenant à ce circuit telle que si  $t_i$  intervient dans  $\bar{t} \in Sync$  alors  $\bar{t} \in S'$ .

Naturellement, tout réseau d'automates temporisés possède au moins un sous-ensemble inévitable de transitions, l'ensemble des transitions lui-même. Cependant, l'efficacité de la méthode va dépendre de deux caractéristiques de cet ensemble : sa taille et son facteur de synchronisation<sup>11</sup>, i.e.  $|I(\bar{t})|$ .

**Remarque 7.52.** Nous pourrions exiger dans la définition précédente que ces transitions apparaissent sur les cycles de  $\mathcal{A}$  et non sur les cycles des  $\mathcal{A}_i$ . Cette modification permet d'obtenir des préfixes plus petits. Cependant, ceci nécessite de calculer le graphe sous-jacent de  $\mathcal{A}$ .  $\lrcorner$

Nous transformons à présent le réseau  $\mathcal{A}$  de sorte que lorsqu'une transition synchronisée est franchie, celle-ci synchronise l'ensemble des processus.

**Définition 7.53** (Événements synchronisés). Soit  $\mathcal{A} = (\mathcal{A}_i)_{1 \leq i \leq n}$  un réseau d'automates temporisés et  $S'$  un sous-ensemble inévitable des transitions synchronisées de  $\mathcal{A}$ , alors

<sup>11</sup>Rappelons que  $I(\bar{t})$ , défini page 193, donne l'ensemble des indices mis en jeu dans  $\bar{t}$ .

- si  $\bar{t} \in S'$ , l'ensemble de ses « copies synchronisées » est  $\text{Sync}(\bar{t}) = \{\bar{f} \mid \forall i \in I(\bar{t}), f_i = t_i \text{ et } \forall i \notin I(\bar{t}), \exists l_i \in L_i \text{ tel que } f_i = l_i \xrightarrow{\text{true}, \varepsilon, \emptyset} l_i\}$ .
- $\mathcal{A}(S')$  est le réseau d'automates temporisés dans lequel l'ensemble de transitions  $S'$  est remplacé par l'ensemble  $\bigcup_{\bar{t} \in S'} \text{Sync}(\bar{t})$ .

Notons que  $\mathcal{A}(S')$  n'est pas définie *via* une fonction de synchronisation mais directement par l'ensemble de ses transitions. Cependant, les résultats précédents s'appliquent également sur des réseaux d'automates temporisés définis ainsi. Notons également que  $\mathcal{A}$  et  $\mathcal{A}(S')$  possèdent le même ensemble de séquences temporisées finies ou infinies avec les mêmes configurations intermédiaires. En particulier, toute propriété exprimée par rapport à ces séquences temporisées étendues est équivalente pour  $\mathcal{A}$  et  $\mathcal{A}(S')$ . C'est le cas pour l'accessibilité et l'occurrence d'un événement qui sont les propriétés auxquelles nous nous intéressons ici. Enfin, dans le cas favorable où chaque élément de  $S'$  est déjà une transition synchronisant l'ensemble des processus, *i.e.* dans le cas où  $I(\bar{t}) = \{1, \dots, n\}$  pour tout  $\bar{t} \in S'$ , alors nous avons  $\mathcal{A}(S') = \mathcal{A}$ .

Les événements synchronisés vérifient les propriétés suivantes.

**Lemme 7.54** (Propriétés des événements synchronisés). *Soit  $t$  un événement synchronisé et  $C'$  une configuration vérifiant  $t \in C'$ . Les propriétés suivantes sont vérifiées :*

- (i)  $\text{Prim}(t) = \{[t]\}$ ,
- (ii)  $[t] \sqsubseteq C'$ .

La propriété (i) entraîne qu'il existe une unique zone relative associée à un événement synchronisé  $t$ , la zone Relativisée( $[t]$ ). Dans la suite, nous noterons cette zone Relativisée( $t$ ). La propriété (ii) est à rapprocher du lemme 7.41.

**Preuve.** Considérons une configuration  $C'$  contenant l'événement  $t$ . Puisque  $t$  est synchronisé, il consomme toutes les conditions correspondant à des états de contrôle. De plus, par définition du dépliage, tout événement du dépliage consomme au moins une condition correspondant à un état de contrôle. Ceci impose que tout autre événement  $t' \in C'$  est ordonné par rapport à  $t$  : nous avons ou bien  $t' < t$ , ou bien  $t' > t$ . Ceci conclut la preuve de ce lemme.  $\square$

Pour comparer les événements synchronisés, nous adaptons la notion d'ordre adéquat introduite dans [ERV02] afin d'obtenir des préfixes plus concis.

**Définition 7.55** (Ordre adéquat, adapté de [ERV02, Définition 4.5]). *Un ordre partiel  $\triangleleft$  sur les configurations finies du dépliage d'un réseau d'automates temporisés est un ordre adéquat s'il vérifie les propriétés suivantes :*

- $\triangleleft$  est bien fondé,
- $C_1 \sqsubseteq C_2$  implique  $C_1 \triangleleft C_2$ ,
- $\triangleleft$  est préservé par les extensions finies : si  $C_1 \triangleleft C_2$  et  $\mu(\text{Cut}(C_1)) = \mu(\text{Cut}(C_2))$ , alors pour toute extension finie  $C_1 \oplus E$  de  $C_1$ , il existe une extension isomorphe  $C_2 \oplus E'$  de  $C_2$ , et réciproquement.

**Exemple 7.56** (Ordres adéquats). L'exemple le plus simple d'ordre adéquat est le suivant :

$$C_1 \triangleleft C_2 \iff C_1 \sqsubseteq C_2$$

Un exemple plus élaboré est :

$$C_1 \triangleleft C_2 \iff |C_1| < |C_2|$$

D'autres exemples sont étudiés dans [ERV02].  $\lrcorner$

Nous définissons la notion de redondance considérée dans le cadre des événements synchronisés. Celle-ci diffère de la notion considérée dans le cadre des automates bornés.

**Définition 7.57** (Événement synchronisé redondant). *Soit  $\mathcal{A}$  un réseau d'automates temporisés et  $S'$  un ensemble de transitions inévitables. Notons  $K$  la constante maximale apparaissant dans le réseau  $\mathcal{A}$ . Considérons le dépliage  $\text{Dépliage}(\mathcal{A}(S'))$  du réseau d'automates temporisés  $\mathcal{A}(S')$  et un ordre adéquat  $\triangleleft$ . Un événement synchronisé  $t$  est dit redondant s'il existe un événement synchronisé  $t'$  dans  $\text{Dépliage}$  vérifiant les conditions suivantes :*

$$\begin{cases} t' \triangleleft t \\ \mu(\text{Cut}(t')) = \mu(\text{Cut}(t)) \\ \text{Approx}_K(\text{Relativisée}(t)) \subseteq \text{Approx}_K(\text{Relativisée}(t')) \end{cases} \quad (7.4)$$

Nous dirons alors que  $t$  est redondant vis-à-vis de  $t'$ .

Les différences avec la définition 7.45 introduite dans le cadre des automates bornés sont les suivantes. D'abord, la définition se rapproche des définitions utilisées dans le cadre des réseaux de Petri sans arcs de lecture. En effet, nous pouvons définir directement les événements redondants sans considérer l'ensemble des configurations primitives associées à cet événement. Ceci provient de la propriété (i) énoncée dans le lemme 7.54. Poursuivant dans cette voie, cette définition fait intervenir la notion d'ordre adéquat rappelée plus haut. Par ailleurs, le réseau d'automates temporisés n'étant plus borné, nous utilisons l'opérateur d'extrapolation  $\text{Approx}_K$ , introduit dans le cadre de l'analyse en avant des automates temporisés dans le chapitre 6 page 154. Cet opérateur permet d'abstraire les valeurs trop grandes des horloges. Nous allons démontrer pourquoi cette abstraction est correcte dans le cadre des événements synchronisés mais ne l'est pas dans le cas général. Avant cela, nous introduisons une notion de  $K$ -approximation sur les valuations d'horloges.

**Définition 7.58.** *Étant données deux valuations  $v$  et  $v'$  définies sur un ensemble d'horloges  $X$ , nous définissons la relation  $\approx_K$  par :*

$$v \approx_K v' \iff \forall x \in X, \begin{cases} v(x) = v'(x) & \text{si } |v(x)| \leq K, \\ |v'(x)| > K & \text{sinon.} \end{cases}$$

*Intuitivement, nous avons la relation  $v \approx_K v'$  si et seulement si pour toute horloge  $x \in X$ , les valeurs  $v(x)$  et  $v'(x)$  sont simultanément inférieures ou strictement supérieures à  $K$  et si elles sont inférieures à  $K$ , alors elles sont égales.*

*Étant données deux configurations  $(\ell, \nu)$  et  $(\ell', \nu')$  d'un réseau d'automates temporisés, nous étendons la notation précédente en écrivant  $(\ell, \nu) \approx_K (\ell', \nu')$  si et seulement si  $\ell = \ell'$  et  $\nu \approx_K \nu'$ .*

Ceci correspond à l'idée que seules les valeurs inférieures à la constante d'extrapolation  $K$  sont importantes. Les valeurs supérieures peuvent être abstraites. De plus, il est facile de vérifier que deux configurations en relation  $\approx_K$  sont équivalentes du point de vue de la bisimulation forte [BBFL03].

**Lemme 7.59** (Extension). *Soit  $(\beta, v)$  un processus non branchant temporisé cohérent  $(\beta, v)$  tel que la configuration  $[\beta]$  contienne un événement synchronisé  $t$  redondant vis-à-vis de  $t'$ . Le point (ii) du lemme 7.54 implique que  $[t] \sqsubseteq [\beta]$ . Notons  $[\beta] = [t] \oplus Y$  cette extension.*

*Alors il existe un processus non branchant temporisé cohérent  $(\beta', v')$  vérifiant les propriétés suivantes :*

$$\begin{cases} [\beta'] = [t'] \oplus Y' \\ Y' \text{ et } Y \text{ isomorphes via } \mu \\ \text{Config}(\beta', v') \approx_K \text{Config}(\beta, v) \end{cases} \quad (7.5)$$

**Preuve.** La preuve suit les lignes de la preuve du lemme 7.46. Notons  $(\beta_1, v_1)$  la restriction de  $(\beta, v)$  à la configuration  $[t]$ . Puisque  $(\beta, v)$  est cohérent,  $(\beta_1, v_1)$  l'est également et nous obtenons  $v_1 \in W_{\beta_1}$ . Nous introduisons les notations suivantes :

- $V = V([t])$ ,
- $V' = V([t'])$ ,
- $\eta$  dénote la bijection canonique entre  $V$  et  $V'$ .

L'existence de l'application  $\eta$  est assurée par la propriété  $\mu(\text{Cut}([t])) = \mu(\text{Cut}([t']))$ . Notons  $w_1$  la valuation de  $V$  obtenue à partir de  $v_1$  en « relativisant » par rapport à la date  $v_1(\mathbf{d}_f(t))$ . L'inclusion  $\text{Approx}_K(\text{Relativisée}(t)) \subseteq \text{Approx}_K(\text{Relativisée}(t'))$  entraîne alors l'existence d'une valuation  $w'_1$  de  $V'$  appartenant à  $\text{Relativisée}(t')$  et vérifiant  $w'_1 \circ \eta \approx_K w_1$ . Il est possible « d'étendre » cette valuation en une valuation  $v'_1$  du processus non branchant  $[t']$ . Nous obtenons le processus non branchant temporisé cohérent  $(\beta'_1, v'_1)$  vérifiant les propriétés suivantes :

$$\begin{cases} [\beta'_1] = [t'] \\ v'_1(\mathbf{d}_f(t')) - v'_{1|V'} \circ \eta \approx_K v_1(\mathbf{d}_f(t)) - v_{1|V} \end{cases}$$

où la notation  $v|_V$  représente la restriction de la valuation  $v$  au sous-ensemble de variables  $V$ . En effet, nous avons simplement  $w_1 = v_1(\mathbf{d}_f(t)) - v_{1|V}$  et  $w'_1 = v'_1(\mathbf{d}_f(t')) - v'_{1|V'}$ . Remarquons alors que comme  $t$  et  $t'$  sont synchronisés, l'approximation symbolisée par  $\approx_K$  est en fait une égalité pour toutes les horloges  $\mathbf{d}_p(p)$  où  $p \in V \cap \mu^{-1}(L)$  puisque dans les zones relativisées ces horloges ont une valeur nulle ! En effet, toutes ces conditions ont été produites lors du franchissement de  $t$  par définition d'un événement synchronisé. Ainsi les approximations (réalisées au-delà de la constante  $K$ ) ne concernent que les horloges du réseau  $\mathcal{A}$ , *i.e.* les horloges liées à  $X$ . De plus, la propriété précédente liant  $v_1$  et  $v'_1$  entraîne également :

$$\text{Config}(\beta_1, v_1) \approx_K \text{Config}(\beta'_1, v'_1)$$

$\text{Config}(\beta_1, v_1)$  et  $\text{Config}(\beta'_1, v'_1)$  sont donc fortement bisimilaires et les configurations atteintes après le franchissement d'une même transition sont encore dans la relation  $\approx_K$ . D'après la remarque 7.23 énonçant que l'extension d'un processus non branchant temporisé cohérent est stable par décalage du temps global, nous pouvons alors démontrer par induction sur la taille de  $Y$  qu'il est possible d'étendre le processus  $(\beta'_1, v'_1)$  en un processus non branchant temporisé cohérent  $(\beta', v')$  vérifiant les propriétés annoncées, ce qui conclut la preuve de ce lemme. □

**Remarque 7.60** (À propos des gardes diagonales et de l'extrapolation). L'utilisation de l'opérateur d'extrapolation n'engendre pas de problèmes de correction dans ce cadre. En effet, une fois la zone relativisée obtenue, nous avons vu que dans le cas d'un événement synchronisé, les seules variables sujettes à l'application de l'extrapolation sont les variables d'horloges correspondant aux horloges du réseau d'automates temporisés, or le réseau d'automates temporisés ne contient pas de gardes diagonales. ┘

**Exemple 7.61** (De l'utilisation de l'opérateur d'extrapolation dans le cas général). Nous illustrons sur un exemple les problèmes posés par l'utilisation de l'opérateur d'extrapolation dans le cas général, *i.e.* pour des événements non synchronisés. Considérons le réseau d'automates temporisés représenté sur la sous-figure 7.11(a) ainsi que le préfixe de son dépliage représenté sur la sous-figure 7.11(b). Les événements  $t_3$  et  $t_5$  vérifient toutes les conditions des événements redondants, hormis le fait d'être

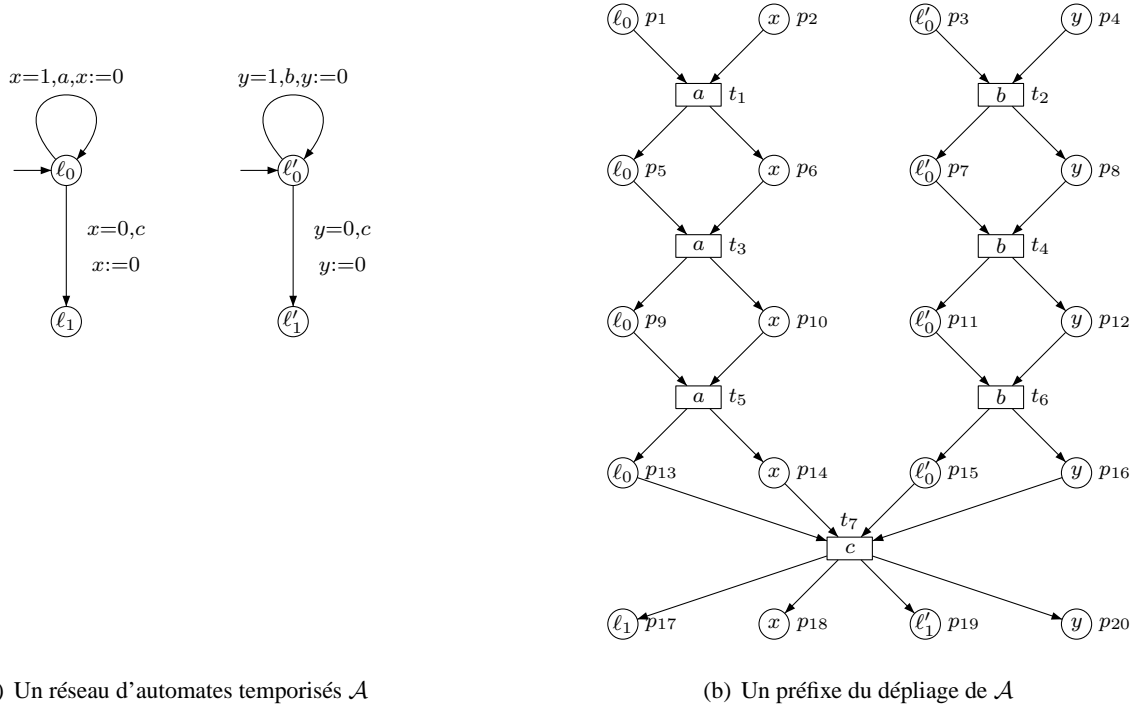


FIG. 7.11 – Contre-exemple à l'utilisation de l'extrapolation.

des événements synchronisés. En effet, leurs coupes contiennent des conditions similaires du point de vue de l'étiquetage et leurs zones relativisées sont décrites par les contraintes d'horloges suivantes :

$$\begin{aligned} \text{Relativisée}(t_3) : \quad & \mathbf{d}_p(p_9) = \mathbf{d}_p(p_{10}) = \mathbf{d}_r(p_{10}) = 0 \\ & \mathbf{d}_p(p_3) = \mathbf{d}_p(p_4) = \mathbf{d}_r(p_4) = 2 \end{aligned}$$

$$\begin{aligned} \text{Relativisée}(t_5) : \quad & \mathbf{d}_p(p_{13}) = \mathbf{d}_p(p_{14}) = \mathbf{d}_r(p_{14}) = 0 \\ & \mathbf{d}_p(p_3) = \mathbf{d}_p(p_4) = \mathbf{d}_r(p_4) = 3 \end{aligned}$$

Nous obtenons donc facilement, en prenant 1 comme constante d'extrapolation et en renommant les horloges de façon adéquate, l'égalité suivante :

$$\text{Approx}_1(\text{Relativisée}(t_3)) = \text{Approx}_1(\text{Relativisée}(t_5))$$

Pourtant, ces deux événements ne possèdent pas les mêmes extensions. Pour le voir, considérons le processus non branchant temporisé  $(\beta, v) = Pr-T(\sigma)$  associé à la séquence temporisée  $\sigma$  suivante :

$$\sigma = (a, 1)(b, 1)(a, 2)(b, 2)(a, 3)(b, 3)(c, 3)$$

Nous avons alors  $[\beta] = \{t_1, \dots, t_7\}$  et  $[t_5] \sqsubseteq [\beta]$ . Il n'existe pourtant pas de processus non branchant temporisé cohérent similaire obtenu comme une extension de  $[t_3]$  par exactement 3 occurrences de  $b$  et une occurrence de  $c$ . En effet, le temps permet ici de compter le nombre d'occurrences de  $a$  et de  $b$ . Ce nombre doit être égal afin de permettre le franchissement de  $c$ . Ainsi, il n'est pas possible de franchir  $c$  après avoir réalisé 2 actions  $a$  et 3 actions  $b$ .  $\lrcorner$

---

**Algorithme 11** Construction d'un préfixe fini et complet du dépliage temporisé à l'aide d'événements synchronisés

---

**Entrée :** Un réseau d'automates temporisés  $\mathcal{A}$ .

**Sortie :** Un préfixe fini et complet  $Préfixe-S(\mathcal{A})$  du dépliage temporisé de  $\mathcal{A}$ .

```

1:  $Préfixe-S := \{(\ell_{1,0}, \emptyset), \dots, (\ell_{n,0}, \emptyset)\} \cup \{(x, \emptyset) \mid x \in X\};$  /* Initialisation */
2:  $ep := EP(Préfixe-S);$  /* Extensions possibles */
3: Tant Que  $ep \neq \emptyset$  Faire
4:   Choisir un événement  $t = (\bar{t}, Y_c, Y_l)$  dans  $ep$ .
5:   Calculer la zone  $W_{[t]}^{loc}$  à l'aide de l'algorithme 8.
6:   Si  $W_{[t]}^{loc} = \emptyset$  Alors /*  $t$  est temporellement incohérent */
7:     Marquer  $t$  comme événement inutile.
8:   Sinon /*  $t$  est temporellement cohérent */
9:     Si  $t$  n'est pas un événement synchronisé Alors
10:       $Préfixe-S := Extension(Préfixe-S, t);$ 
11:       $ep := EP(Préfixe-S);$ 
12:     Sinon /*  $t$  synchronisé */
13:      Si  $t$  est redondant Alors
14:        Marquer  $t$  comme inutile.
15:      Sinon /*  $t$  non redondant */
16:         $Préfixe-S := Extension(Préfixe-S, t);$ 
17:         $ep := EP(Préfixe-S);$ 
18:      Fin Si
19:    Fin Si
20:  Fin Si
21: Fin Tant Que
22: Retourner  $Préfixe-S$ ;

```

---

**Présentation de l'algorithme.** Nous décrivons l'algorithme global associé à la méthode développée dans cette sous-section. Nous notons  $Préfixe-S(\mathcal{A})$  le préfixe<sup>12</sup> de  $Dépliage(\mathcal{A})$  construit par cet algorithme, présenté comme l'algorithme 11. Comme précédemment, l'algorithme procède par extensions successives. Si la zone associée à un événement est vide (ligne 6), celui-ci est déclaré inutile. Dans le cas contraire, les événements synchronisés et les événements non synchronisés sont traités différemment. Ainsi, pour les événements non synchronisés, l'algorithme se contente, comme l'algorithme 6 calculant le dépliage complet, d'étendre le préfixe et de mettre à jour la liste des extensions possibles (lignes 9 à 11). Pour les événements synchronisés, si l'événement est redondant, alors le dépliage est stoppé et l'événement est déclaré inutile (lignes 13, 14). Dans le cas contraire, l'algorithme procède comme plus haut en étendant le préfixe et en mettant à jour la liste des extensions possibles (lignes 15 à 17). Naturellement, les événements déclarés inutiles ne sont pas pris en compte lors du calcul des extensions possibles, *i.e.* ils ne peuvent pas être étendus. Ainsi, les événements redondants correspondent bien à des arrêts du processus de dépliage.

Le théorème suivant établit la terminaison et la correction de l'algorithme 11. Notons qu'ici le préfixe fini construit est complet à  $K$ -approximation près ce qui est inévitable puisque le réseau n'est pas borné.

---

<sup>12</sup>En réalité,  $Préfixe-S(\mathcal{A})$  est un préfixe de  $Dépliage(\mathcal{A}(S'))$ .

**Théorème 7.62.** *L'algorithme 11 termine et le préfixe fini calculé  $\text{Préfixe-S}(\mathcal{A})$  est complet à  $K$ -extrapolation près, i.e. si une configuration  $(\ell, \nu)$  est accessible dans  $\mathcal{A}$ , alors il existe une configuration  $(\ell, \nu')$  représentée dans ce préfixe vérifiant  $\nu \approx_K \nu'$ .*

**Preuve. Terminaison.** Afin de démontrer ce point, nous définissons la notion de chaîne causale. Étant donné un événement  $t$  du dépliage  $\text{Dépliage}(\mathcal{A})$  du réseau  $\mathcal{A}$ , une *chaîne causale* de  $t$  est une séquence d'événements appartenant au passé causal fort de  $t$  dans  $\text{Dépliage}(\mathcal{A})$  et menant à  $t$ . Ceci correspond aux différents entrelacements possibles des événements concurrents du passé causal fort de  $t$ . Rappelons que dans le passé causal fort, certains événements ne sont pas « nécessaires » pour le franchissement de  $t$ . Nous notons  $d(t)$  la longueur maximale d'une chaîne causale de  $t$ , appelée *profondeur* de  $t$ . Cette définition est correcte car le nombre de chaînes causales de  $t$  est fini. Il est facile de vérifier que pour tout entier naturel  $p$ , le nombre d'événements  $t$  de profondeur bornée par  $p$  est fini. En particulier, si l'algorithme ne termine pas, alors il est possible de trouver des chaînes causales de longueur arbitrairement grande. Nous allons démontrer que ceci n'est pas possible.

Utilisant les notations précédentes, nous notons  $\mathcal{A} = ((\mathcal{A}_i)_{1 \leq i \leq n}, f)$  le réseau d'automates temporisés que nous considérons. Notons  $l_c$  la longueur maximale d'un circuit élémentaire apparaissant dans un des automates temporisés  $\mathcal{A}_i$ , pour  $i \in \{1, \dots, n\}$ . Si nous considérons une chaîne causale de longueur strictement supérieure à  $n \times l_c$ , alors cette chaîne contient nécessairement deux occurrences d'événements synchronisés, puisque tout circuit élémentaire contient au moins une occurrence d'événement synchronisé. Notons alors  $N$  le produit du nombre de valeurs possibles pour  $\mu(\mathcal{C})$  où  $\mathcal{C}$  décrit l'ensemble des coupes (i.e. simplement le nombre  $\prod_i |L_i|$ ) par le nombre de zones relativisées extrapolées. Ce dernier nombre est fini grâce à l'application de l'opérateur d'extrapolation (rappelons que les zones  $K$ -bornées sont en nombre fini) et car le nombre de variables impliquées dans les zones relativisées est constant et donc borné. Considérons une séquence d'événements de longueur strictement plus grande que  $N \times n \times l_c$ . Alors cette séquence contient deux occurrences  $t_1$  et  $t_2$  d'événements synchronisés dont les coupes ont la même image par  $\mu$  et dont les zones relativisées ont la même image par  $\text{Approx}_K$ . Puisque le second, disons  $t_2$ , apparaît après le premier, disons  $t_1$ , le long de cette séquence, la configuration  $[t_2]$  étend la configuration  $[t_1]$  : nous avons  $[t_1] \sqsubseteq [t_2]$ . En effet,  $t_1$  et  $t_2$  sont des événements synchronisés et le résultat découle donc du lemme 7.54. D'après la seconde propriété des ordres adéquats (définition 7.55), ceci implique  $t_1 \triangleleft t_2$ . Ainsi, nous obtenons que  $t_2$  est redondant vis-à-vis de  $t_1$ . Finalement, le calcul du préfixe  $\text{Préfixe-S}(\mathcal{A})$  par l'algorithme 11 est donc stoppé au-delà de  $t_2$  et la longueur des chaînes causales est donc bornée par  $N \times n \times l_c + 1$ , ce qui conclut la démonstration de ce point.

**Complétude.** Soit  $(\ell, \nu)$  une configuration accessible de  $\mathcal{A}$ . Il existe donc une séquence temporisée  $\sigma$  menant à cette configuration et le processus non branchant temporisé  $(\beta, v) = \text{Pr-T}(\sigma)$  vérifie  $\text{Config}(\beta, v) = (\ell, \nu)$ . Deux cas peuvent alors se produire : ou bien la configuration  $[\beta]$  appartient au préfixe que nous construisons, ou bien ce n'est pas le cas. Dans le premier cas, nous obtenons le résultat annoncé. Dans le second cas, nous allons voir que  $[\beta]$  contient nécessairement un événement redondant. En effet, puisqu'il existe un processus non branchant temporisé cohérent le long de  $\beta$  (le processus  $(\beta, v)$ ), les zones calculées le long de  $\beta$  ne peuvent pas être vides. L'unique origine possible de l'interruption de l'extension de  $\beta$  est donc que  $\beta$  contienne un événement redondant  $t$ . Fixons un tel événement  $t$  et notons  $t'$  un événement vis-à-vis duquel il est redondant. Nous appliquons alors le lemme 7.59 à  $\beta$ ,  $t$  et  $t'$ . Nous obtenons ainsi un nouveau processus non branchant temporisé  $(\beta', v')$  cohérent vérifiant les conditions (7.5). En particulier, nous avons donc  $\text{Config}(\beta', v') \approx_K \text{Config}(\beta, v) = (\ell, \nu)$ . De plus, par construction et car l'ordre adéquat  $\triangleleft$  est stable par extension finie, nous avons également  $[\beta'] \triangleleft [\beta]$ . Nous pouvons alors appliquer à nouveau ce raisonnement à partir de  $(\beta', v')$ . Puisque l'ordre adéquat  $\triangleleft$  est bien fondé, ce processus termine et nous obtenons



finalement un processus non branchant temporisé  $(\beta'', v'')$  tel que la configuration  $[\beta'']$  appartient au préfixe construit et tel que  $Config(\beta'', v'') \approx_K (\ell, \nu)$ , ce qu'il fallait démontrer.  $\square$

## 7.6 Discussion

L'ensemble des techniques développées dans ce chapitre constitue un premier travail dans une direction relativement nouvelle pour les réseaux d'automates temporisés. Le modèle que nous avons considéré est riche (horloges partagées, invariants) et les objets que nous avons construits sont relativement complexes : arcs de lecture, zones, nombre de variables assez important. Il est donc naturel de penser que nos constructions peuvent se simplifier en se restreignant à des sous-classes. Nous discutons cet aspect dans cette section et mettons notre approche en relation avec certains travaux récents.

**Réseaux d'automates temporisés sans horloges partagées et sans invariants.** L'ajout dans la structure de places pour représenter explicitement les horloges et d'arcs de lecture peut paraître arbitraire. Afin de s'affranchir de ces objets, il est intéressant de considérer la classe des réseaux d'automates temporisés sans horloges partagées et sans invariants. Il n'est alors plus nécessaire de représenter les horloges explicitement par des conditions, puisque celles-ci ne peuvent être modifiées que par un seul automate. Étant donnée la condition représentant l'état de contrôle de l'automate « possédant » l'horloge, il est possible de déterminer la dernière transition ayant remis l'horloge à zéro. De même, le retrait des invariants permet de ne plus recourir aux arcs de lecture. En effet, ceux-ci sont utilisés dans notre construction à deux fins. La première consiste à représenter les tests d'horloges qui ne sont pas remis à zéro. Ceci n'est donc plus utile si les horloges ne sont plus représentées. La seconde consiste à exprimer les dépendances vis-à-vis des invariants. Ainsi, pour la sous-classe considérée ici, il est possible de définir un dépliage temporisé du réseau d'automates temporisés comme une extension du dépliage du réseau d'automates finis associé. Les zones attachées aux événements sont sensiblement les mêmes que celles définies dans la section 7.4, excepté qu'il faut traiter les dates de remises à zéro des horloges légèrement différemment.

La deuxième étape, une fois ce dépliage temporisé défini, consiste à étudier la possibilité d'un calcul utilisant des zones locales, comme cela est présenté dans la sous-section 7.4.4. La même difficulté survient alors, *i.e.* la conjonction des zones locales de deux événements concurrents ne correspond pas à la zone locale associée à l'union de ces deux événements. Le contre-exemple 7.30 est en effet fondé sur un réseau d'automates temporisés sans horloges partagées ni invariants. La solution présentée dans cette partie peut être adaptée de sorte à s'appliquer dans ce cadre. Les difficultés sont en fait moindres puisque le réseau ne contient pas d'arcs de lecture et l'algorithme 8 peut être simplifié.

La troisième étape enfin consiste à calculer un préfixe fini et complet de ce dépliage. Remarquons à nouveau que les problèmes rencontrés dans notre cadre sont encore présents pour cette sous-classe. Ainsi, les problèmes liés à l'application de l'extrapolation peuvent encore survenir. En effet, le contre-exemple 7.61 ne contient ni horloges partagées ni invariants. Les restrictions faites sur le modèle ne simplifient donc pas le calcul d'un préfixe fini et complet. Il est toutefois possible d'appliquer la construction fondée sur les événements synchronisés développée dans la sous-section 7.5.3. L'application de la méthode fondée sur le caractère borné des automates ne semble en revanche *a priori* pas possible dans ce cadre puisque la construction ne prend pas en compte les invariants.

En conclusion, la restriction du modèle à la sous-classe des automates temporisés sans horloges partagées et sans invariants conduit donc à une structure plus simple, mais ne permet pas d'éviter les problèmes rencontrés pour le développement d'un algorithme local ni ceux rencontrés pour le calcul d'un préfixe fini et complet.



(a) Un réseau d'automates temporisés  $\mathcal{A}$  avec horloges partagées

(b) Deux zones obtenues lors de l'analyse de  $\mathcal{A}$

FIG. 7.12 – Contre-exemple au théorème 7.63 si les horloges sont partagées.

**Réseaux d'automates temporisés sans horloges partagées : un résultat de convexité.** Des travaux récents [BSBM06] ont démontré un résultat de convexité concernant les réseaux d'automates temporisés sans horloges partagées. Nous détaillons ici ce résultat et comment il se compare à nos travaux.

**Théorème 7.63** (Convexité [BSBM06]). *Soit  $\mathcal{A}$  un réseau d'automates temporisés avec invariants et sans horloges partagées. Considérons un ensemble de transitions  $T$  du réseau  $\mathcal{A}$ , deux états de contrôles globaux  $\ell$  et  $\ell'$  de  $\mathcal{A}$  et une zone  $Z$  portant sur l'ensemble d'horloges  $X$  de  $\mathcal{A}$ . L'ensemble suivant est alors une zone sur  $X$ .*

$$R_{Z,T} = \bigcup_{\sigma \in \text{exec}(T)} \{v' \mid \exists v \in Z, (\ell, v) \xrightarrow{\sigma} (\ell', v')\}$$

où  $\text{exec}(T)$  représente l'ensemble des entrelacements possibles de transitions de  $T$ .

Intuitivement, ceci signifie que l'union de tous les comportements possibles du réseau exécutant le même ensemble  $T$  de transitions constitue une zone. Ce résultat est utilisé afin de rassembler en une unique zone les zones calculées le long de chacun de ces entrelacements lors de l'analyse en avant de  $\mathcal{A}$ , obtenant ainsi des résultats très encourageants. Nous allons dans un premier temps montrer que ce résultat n'est pas valable pour le modèle des réseaux d'automates temporisés avec invariants et horloges partagées.

Le théorème 7.63 repose en effet fortement sur le fait qu'aucune horloge ne soit partagée. Considérons le réseau d'automates temporisés représenté sur la sous-figure 7.12(a). Ce réseau possède deux horloges,  $x$  et  $y$ . L'horloge  $y$  n'est utilisée que par le premier automate tandis que  $x$  est partagée entre les deux automates. Il est facile de calculer la zone atteinte depuis la configuration initiale à l'issue du franchissement des transitions  $a$  et  $b$  dans cet ordre :

$$Z_{a,b} = x \geq 3 \wedge y \geq 3 \wedge 0 \leq y - x \leq 3$$

De même, nous pouvons calculer la zone atteinte depuis la configuration initiale à l'issue du franchissement des transitions  $b$  et  $a$  dans cet ordre :

$$Z_{b,a} = x \geq 0 \wedge y \geq 3 \wedge y - x = 3$$

Ces deux zones sont représentées sur la sous-figure 7.12(b) et il est facile de constater que leur union ne constitue pas une zone, ce qui démontre que le théorème 7.63 n'est pas valable pour le modèle

que nous avons considéré dans notre approche. En particulier, ceci justifie la duplication de certains événements : nous devons distinguer certains entrelacements lorsque, réunis, ils ne permettent pas d'obtenir une zone. L'introduction de places dédiées aux horloges, motivée dans la section 7.2 à l'aide de l'exemple 7.3, a permis de créer ces duplications, qui sont donc présentes dans le dépliage obtenu par notre approche.

Si les horloges ne sont pas partagées, nos résultats permettent d'obtenir une nouvelle preuve du théorème 7.63 si l'ensemble admet une décomposition en tranches. En effet, celui-ci peut être obtenu comme un corollaire de la proposition 7.24 : étant donné le processus non branchant  $\beta$  associé à un ensemble  $T$  de transitions, la zone  $W_\beta$  décrit toutes les contraintes temporisées associées à  $\beta$  ce qui implique que l'ensemble des valuations atteintes à l'issue du franchissement de  $\beta$  constitue une zone. Pour un ensemble  $T$  quelconque, notre algorithme distingue les entrelacements vis-à-vis des modifications d'invariants. Ceci est nécessaire afin d'obtenir des états locaux cohérents temporellement, *i.e.* correspondant à une date absolue que tous les automates peuvent atteindre sans franchir de transitions. Nous discutons cette notion de blocages temporels plus bas.

Notons enfin que notre construction tire profit de la remarque faite par [BSBM06] d'une façon plus efficace. En effet, en plus de fusionner les zones en une seule zone, ce que fait naturellement notre méthode lors du calcul de la zone associée au processus non branchant représentant l'union des différents entrelacements, nous calculons cette zone « globale » beaucoup plus efficacement. Alors que [BSBM06] calcule la zone associée à chaque entrelacement individuellement puis calcule l'union de ces zones, notre méthode permet d'étendre peu à peu le processus dans son ensemble et calcule donc directement la zone globale.

**Réseaux d'automates temporisés sans blocages temporels.** Une autre alternative permettant de réduire les difficultés dues aux invariants consiste à supposer qu'il n'existe pas de blocages temporels dans le réseau d'automates temporisés. Ceci signifie qu'une transition de délai locale à un sous-ensemble  $S$  d'automates ne peut pas être empêchée par un automate du réseau n'appartenant pas à  $S$ . En effet, avec notre modèle, nous avons vu dans l'exemple 7.4 qu'un invariant porté par un automate n'intervenant pas dans une transition synchronisée peut malgré tout empêcher son franchissement<sup>13</sup>. L'absence de blocage temporel permet donc d'éliminer ce type de situations. Cette hypothèse a été faite pour la première fois dans [BJLY98]. En permettant de ne pas avoir à considérer les invariants portant sur les autres automates lors de l'analyse des successeurs par une transition donnée, elle simplifie donc fortement l'analyse du réseau.

Cependant, c'est une hypothèse difficile à garantir. Du point de vue syntaxique d'abord, la restriction la plus simple permettant d'assurer la propriété d'absence de blocage temporel consiste à supposer que depuis tout état de contrôle il existe une transition locale, *i.e.* synchronisée avec aucun autre automate du réseau, qui permet à l'automate de quitter cet état et de laisser s'écouler du temps. Cette restriction paraît néfaste pour la modélisation d'un système réel. Du point de vue sémantique, décider si un système présente ou non un blocage temporel est un problème aussi difficile que l'accessibilité. Il n'est donc pas envisageable de chercher, lors d'une première phase de calcul, à résoudre ce problème.

D'autre part, lors de la construction d'un dépliage à l'aide de cette hypothèse, il semble naturel de déplier localement les processus puisque cela est rendu possible. Pourtant, les processus alors obtenus peuvent être incomplets. Plus précisément, il est possible qu'une coupe du dépliage ne corresponde pas à une configuration réelle du réseau. En effet, puisque les échelles de temps sont désynchronisées afin de permettre l'écoulement de temps local, il est possible que certains automates « prennent du

<sup>13</sup>Cette situation peut survenir même sans horloges partagées.

retard ». La propriété garantie par le dépliage assure donc seulement qu'il est possible de prolonger le processus non branchant de sorte à obtenir une exécution du système.

Finalement, il nous semble donc que cette restriction est trop forte et qu'elle conduit à des dépliages non conformes à l'intuition. Par ailleurs, nous soulignons que notre algorithme peut également être utilisé pour la détection de ces blocages temporels.

**Une approche issue des réseaux de Petri temporels.** D'autres travaux ont été réalisés indépendamment et simultanément dans [CCJ06] ayant sensiblement le même objectif : construire un dépliage temporisé d'un réseau d'automates temporisés. Ces travaux présentent certaines similitudes avec les nôtres, mais aussi des points de divergence.

Au niveau du modèle considéré, les réseaux d'automates temporisés considérés dans [CCJ06] possèdent comme dans nos travaux des invariants mais en revanche n'autorisent pas les horloges partagées et requièrent l'hypothèse d'absence de blocages temporels discutée précédemment.

Concernant l'approche algorithmique, ces travaux se placent dans la suite des travaux de [CJ06] pour les réseaux de Petri temporels. Des arcs de lecture sont donc également utilisés afin de dupliquer certains événements n'ayant pas les mêmes conditions temporelles de franchissement, ce qui constitue un point commun important avec notre approche. Cependant, ces arcs de lecture sont ajoutés au dépliage au cas par cas, après un certain nombre de calculs sur les zones locales, contrairement à notre approche qui utilise des critères syntaxiques pour décider *a priori* de la construction du dépliage. L'algorithme global obtenu est donc beaucoup plus coûteux, ce qui le rend inapplicable en pratique mais peut permettre de définir un dépliage plus concis. L'objectif de ces travaux était en fait de définir un dépliage reflétant dans sa structure non temporisée les dépendances causales entre les événements, sans se poser la question de l'efficacité de la construction. Notre approche au contraire cherche à proposer des algorithmes efficaces, mais peut parfois introduire une certaine redondance entre des événements.

## 7.7 Conclusion

Nous avons étudié dans ce chapitre l'application de la méthode des dépliages aux réseaux d'automates temporisés avec invariants et horloges partagées dans le but de développer un nouvel algorithme efficace de vérification. Nous avons défini le dépliage non temporisé associé à un tel réseau, en exprimant explicitement les horloges à l'aide de conditions et en ajoutant des arcs de lecture. Ceci permet de conserver un maximum de concurrence et de localité pour les événements du dépliage. Nous avons ensuite attaché des zones aux événements du dépliage, définissant ainsi un dépliage temporisé. Nous avons proposé un algorithme permettant de construire ce dépliage à l'aide de zones locales de dimension constante. De plus, nous avons présenté deux algorithmes pour la construction d'un préfixe fini de ce dépliage, l'un fondé sur l'algorithme de [Win02], l'autre, original, fondé sur la notion d'événement synchronisé. Nous avons finalement discuté l'adaptation de notre algorithme à des sous-classes du modèle que nous avons considéré.

Les extensions de ces travaux sont diverses. D'abord, plusieurs directions apparaissent suite à la discussion menée dans la section précédente. Ainsi, il serait intéressant de développer plus précisément la construction d'un dépliage pour un modèle sans horloges partagées en évitant la duplication d'événements, ce qui pourrait permettre d'obtenir des objets plus concis. D'autre part, la comparaison de nos travaux avec ceux réalisés dans [CCJ06] indique qu'il est sans doute possible de trouver un compromis entre leur approche et la nôtre. Nous avons déjà travaillé dans cette voie et obtenu diverses caractérisations locales peu coûteuses permettant d'éliminer des duplications superflues. Ensuite, dif-

---

férentes autres pistes peuvent être étudiées afin d'améliorer notre algorithme dans le cas général. Il est par exemple possible de réduire la dépendance aux invariants en faisant une analyse non temporelle du rôle des invariants. En effet, certaines compositions ne peuvent pas survenir et donc certains invariants ne peuvent pas avoir d'impact sur des parties « isolées » du réseau. Cette analyse est à rapprocher de l'approche des horloges actives développées dans [DT98]. Une autre optimisation consiste à réduire le nombre d'horloges utilisées dans les définitions des zones que nous avons attachées aux événements. Il est en effet assez facile de remarquer que les zones que nous avons définies peuvent présenter de la redondance. Par ailleurs, le préfixe que nous avons défini est utilisé ici pour décider des propriétés d'accessibilité et de possibilité de tir d'une transition. Il serait intéressant de développer l'ensemble des propriétés vérifiables à l'aide de ce préfixe. Enfin, un travail majeur à mener à l'issue de nos travaux est le développement d'un outil implémentant les algorithmes présentés dans ce chapitre. Ceci permettra de comparer entre elles et de façon effective les différentes alternatives que nous avons proposées (par exemple la construction avec ou sans zones locales) et surtout de comparer l'efficacité de notre approche avec les algorithmes standards utilisés dans les outils comme UPPAAL ou KRONOS.



## **Quatrième partie**

# **De la vérification du modèle à l'implémentation du système**





## Chapitre 8

# Implémentabilité et robustesse

Dans ce chapitre, nous nous intéressons au problème suivant : une fois démontré qu'un modèle vérifie une propriété, comment s'assurer qu'il existe une implémentation de ce modèle qui vérifie également cette propriété et comment synthétiser un tel système ? Les travaux présentés dans ce chapitre ont été publiés dans [BMR06].

### 8.1 Introduction

**Motivations.** Les travaux présentés dans les trois premières parties se situent dans le cadre de la vérification formelle. Ce domaine de recherche est devenu très actif dans l'informatique théorique au cours des trente dernières années. Plus récemment la problématique du contrôle a acquis une place importante dans ce milieu. L'importance de la prise en compte de contraintes temps-réel quantitatives dans ces problèmes a rapidement été reconnue par la communauté scientifique et des modèles temporisés sont apparus. Nous avons déjà présentés plusieurs de ces modèles dans la partie I et les avons étudiés et comparés dans la partie II. De nombreux algorithmes adaptés à ces modèles ont été conçus (nous y avons contribué dans la partie III) et ont permis le développement d'outils de vérification largement utilisés comme UPPAAL et KRONOS pour les automates temporisés et TINA et ROMEO pour les réseaux de Petri temporels.

Malgré l'étendue des résultats théoriques sur la vérification des systèmes temporisés, l'implémentation de ces systèmes constitue un problème délicat. En effet, les automates temporisés, comme les extensions temporisées des réseaux de Petri, sont gouvernés par des sémantiques théoriques et idéalisées tandis que leur implémentation sur des matériaux informatiques réels est contrôlée par des lois physiques. Ces deux types de contraintes ne sont pas toujours compatibles et les implémentations des modèles temporisés que nous avons considérés peuvent être sensiblement différentes. En particulier, l'implémentation d'un modèle peut ne pas vérifier une propriété, même si le modèle la vérifie. Ceci constitue un défaut important que nous allons chercher à résoudre dans ce chapitre.

Les incompatibilités entre les sémantiques théoriques des modèles temporisés et les lois physiques vérifiées par les systèmes réels sont multiples. D'abord, les modèles considérés supposent que les horloges manipulées sont continues et infiniment précises, alors que les processeurs ont une fréquence finie et sont digitaux. Ainsi, les comportements Zeno que nous avons déjà mis en évidence dans les parties précédentes et qui peuvent être acceptés par les systèmes temporisés que nous avons considérés ne peuvent pas être réalisés par un système réel. Cependant, ce ne sont pas les seuls et, par exemple, [CHR02] présente un automate temporisé réalisant des actions discrètes aux dates  $n$  et  $n + \frac{1}{n}$  pour tout entier naturel  $n$ . À nouveau, aucune implémentation d'un tel automate ne pourra pas vérifier cette

propriété car celui-ci est infiniment rapide. Une autre différence entre les deux domaines considérés provient de l'aspect immédiat des communications. Ceci est en particulier le cas dans la synthèse de contrôleurs, domaine dans lequel l'objectif est la construction d'un contrôleur (par exemple sous la forme d'un automate temporisé) interagissant avec l'environnement pour piloter un système afin de satisfaire une certaine propriété. Dans les réseaux d'automates temporisés, les synchronisations sont supposées immédiates ce qui n'est pas réaliste en pratique.

Afin de prendre en compte ces difficultés, [DDR04] a introduit une nouvelle sémantique pour les automates temporisés, intitulée sémantique « AASAP » (AASAP signifie « Almost As Soon As Possible » en anglais, *i.e.* « presque dès que possible »), autorisant des comportements supplémentaires. Celle-ci intègre à la fois la dimension digitale des composants électroniques ainsi que l'aspect non immédiat des communications. Les travaux [DDR04] montrent également que cette nouvelle sémantique est proche de la sémantique réelle d'une implémentation, prouvant même que la vérification d'une propriété pour la sémantique AASAP assure l'existence d'une implémentation vérifiant cette propriété. Ceci ramène alors le problème de l'implémentation au problème de la vérification d'une propriété par un automate pour cette nouvelle sémantique, problème que nous appelons « vérification robuste », puisque la propriété doit être satisfaite pour des comportements supplémentaires obtenus en introduisant des imprécisions dans le modèle. Ce problème a été résolu pour des propriétés d'accessibilité dans [DDMR04a, Pur98] et nous allons étendre dans ce chapitre le type de propriétés pour lesquelles le problème de vérification robuste est décidable. Enfin, dans [DDR05b, DW07], cette démarche est automatisée dans un outil permettant la génération automatique de code, et celui-ci est appliqué sur l'exemple du protocole Philips pour le contrôle audio.

### Travaux existants.

- L'approche que nous avons décrite ci-dessus contraste avec d'autres solutions fondées sur une modélisation explicite de l'ensemble de la plate-forme d'exécution de l'automate temporisé, comme dans [AT05]. Ceci permet d'obtenir un cadre de travail très riche, permettant par exemple de faire différents choix de fonctionnement pour le processeur, ou pour le traitement des synchronisations. Cependant, cette approche souffre de deux défauts : d'abord, elle ne vérifie pas l'hypothèse « plus c'est rapide, mieux c'est », vérifiée par la sémantique AASAP, qui exprime que si un automate peut être implémenté sur une certaine architecture de façon « correcte », alors c'est également le cas pour une architecture plus rapide. De plus, les problèmes de vérification sont plus difficiles et les travaux [AT05] ne présentent pas de résultats de décidabilité.
- Une notion d'automates temporisés « robustes » a été proposée et étudiée dans [GHJ97, HR00, OW03]. Cependant, celle-ci ne correspond pas à la sémantique AASAP. En effet, dans ces travaux, pour qu'une exécution de l'automate temporisé soit acceptante, il faut que toutes les exécutions proches, du point de vue topologique, le soient pour la sémantique classique. Par conséquent, leur approche peut restreindre le langage accepté tandis que la sémantique AASAP l'élargit toujours.
- Dans [HMP05], une notion de similarité entre des systèmes de transitions temporisés a été introduite. Dans ces travaux, les auteurs montrent que deux systèmes proches vérifient des formules de TCTL proches, ce qu'ils présentent comme une propriété de robustesse de TCTL. Celle-ci est donc différente de la satisfaction robuste que nous considérons et leur notion ne permet pas de garantir l'implémentabilité du système.
- Dans [Pur98, ALM05], une déviation des pentes des horloges est autorisée. Une telle perturbation est proche du modèle que nous considérons. En effet, [Pur98] démontre que l'élargissement

des contraintes introduit par la sémantique AASAP est équivalent pour des propriétés d'accessibilité à la déviation des pentes d'horloges.

- Enfin, nos travaux doivent être rapprochés du problème de la génération de code à partir de modèles temporisés. L'outil TIMES [AFM<sup>+</sup>03] génère automatiquement du code exécutable correspondant à un automate temporisé. Cependant, ces travaux ne prennent pas en compte les problèmes liés à l'imprécision des matériels utilisés. Henzinger et al. proposent un modèle pour les logiciels embarqués, intitulé GIOTTO, qui peut être vu comme une étape intermédiaire entre les modèles théoriques comme les automates temporisés et les plate-formes réelles [HHK03].

**Contributions.** Dans ce chapitre, nous nous plaçons dans le cadre de la sémantique AASAP pour les automates temporisés. Nous proposons des algorithmes permettant de résoudre le problème de la vérification robuste de spécifications plus générales que l'accessibilité comme des propriétés de Büchi ou des propriétés exprimées dans la logique LTL. Les algorithmes que nous proposons sont fondés sur une extension de la construction de l'automate des régions issue des remarques faites dans [DDMR04a]. Nos algorithmes demeurent dans la classe de complexité PSPACE et nous démontrons qu'ils sont optimaux. Par ailleurs, nous développons également un algorithme PSPACE pour la vérification de propriétés temporisées simples, comme la propriété de temps de réponse borné. Cet algorithme est *ad-hoc*, mais il constitue une première étape vers la vérification d'un ensemble plus grand de propriétés temporisées.

**Organisation du chapitre.** Dans la section 8.2, nous présentons le cadre de travail de la sémantique AASAP ainsi que les résultats importants, et définissons la notion de vérification robuste pour des propriétés de chemin et présentons l'algorithme permettant de résoudre le cadre des propriétés d'accessibilité. Dans la section 8.3, nous résolvons le problème de la vérification robuste de conditions co-Büchi. Nous appliquons ensuite ces résultats à la vérification de propriétés exprimées en LTL dans la section 8.4. Enfin, nous présentons nos premiers résultats pour les propriétés temporisées dans la section 8.5 et concluons dans la section 8.6.

## 8.2 Vérification et implémentabilité

Nous décrivons dans cette section le cadre de travail dans lequel nous nous plaçons et présentons les résultats existants pour des propriétés d'accessibilité. Avant cela, nous présentons un exemple illustrant certains problèmes posés par la sémantique idéalisée des automates temporisés.

**Exemple 8.1** (Protocole de Fischer). Nous considérons le protocole d'exclusion mutuelle de Fischer [KLL<sup>+</sup>97] : deux systèmes  $S_1$  et  $S_2$  souhaitent accéder à une section critique représentée par l'état SC ; ils sont supervisés par deux contrôleurs  $C_1$  et  $C_2$  chargés d'assurer que les deux systèmes n'accéderont pas simultanément à la section critique. La figure 8.1 représente un système et son contrôleur. Il est facile de montrer, grâce à un outil de vérification d'automates temporisés comme UPPAAL, HYTECH ou KRONOS, que l'exclusion mutuelle est bien réalisée par ces contrôleurs.

La propriété d'exclusion mutuelle de notre exemple est cependant étroitement liée au caractère infiniment précis (*i.e.* exact) du modèle des automates temporisés qui le composent. En particulier, elle repose sur la garde stricte  $x_i > 2$ . Celle-ci assure en effet, combinée avec l'invariant  $x_i \leq 2$  de l'état Requête, que lorsqu'un processus  $P_i$  peut atteindre sa section critique depuis l'état Attente, l'autre processus  $P_{3-i}$  ne peut être dans l'état Requête. Ainsi, relâcher la contrainte  $x_i > 2$  en la contrainte  $x_i \geq 2$  fait perdre la correction du système, ce qui prouve la fragilité du modèle.  $\lrcorner$

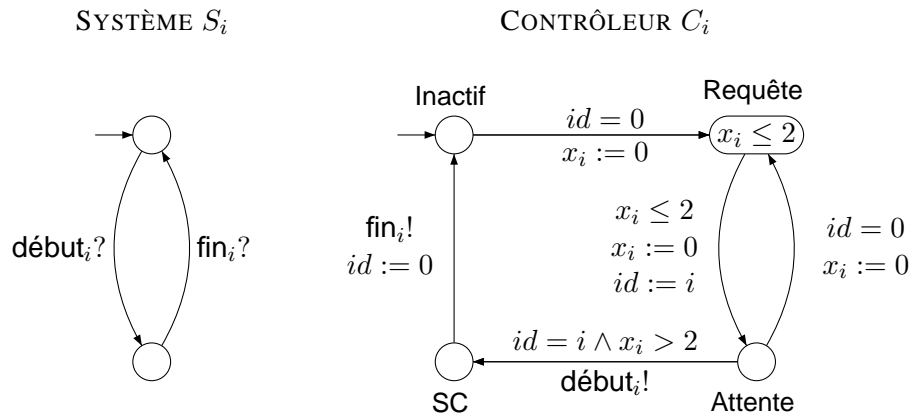


FIG. 8.1 – Le protocole d'exclusion mutuelle de Fischer

### 8.2.1 Cadre de la sémantique « AASAP »

D'après les constatations faites dans l'exemple 8.1 et d'après les définitions de la sémantique d'un automate temporisé et d'un réseau d'automates temporisés (données page 36 et page 47 respectivement), plusieurs aspects ne correspondent pas à l'intuition associée à un système réel. Nous en donnons quelques uns :

- Des transitions peuvent être franchies en temps nul, et en nombre arbitrairement grand.
- Les variables utilisées ont une précision « parfaite », *i.e.* infinie.
- La synchronisation entre les différents automates est réalisée en temps nul.

Naturellement, un système réel, lorsqu'il doit simuler le comportement d'un automate temporisé, ne peut pas décider du franchissement d'une transition en temps nul. De même, les variables qu'il manipule ont une précision finie et les échanges de messages lors des synchronisations entre les différents processus ne peuvent pas être instantanés. L'hypothèse habituelle permettant de s'affranchir de ces problèmes consiste à supposer que l'ordre de grandeur des temps de calcul ou des envois/réceptions de messages est infiniment plus petit que l'échelle de temps manipulée par l'automate dans ses gardes et ses invariants. Cependant, cette approche ne repose sur aucun fondement mathématique, et il est donc nécessaire de pouvoir la démontrer pour transférer les propriétés de correction du modèle construit au système réel.

Afin de prendre en compte les différentes imperfections d'un système réel, une nouvelle sémantique intitulée **AASAP** est introduite pour les automates temporisés. Celle-ci relâche les hypothèses trop fortes imposées par la sémantique classique des automates temporisés. Nous ne la présentons pas formellement ici mais nous donnons l'intuition de son fonctionnement, et ses principales propriétés. Nous renvoyons le lecteur à [DDR05a] pour une présentation complète. La notion de simulation introduite page 34 pour les systèmes de transitions temporisés s'applique directement à ce cadre. Celle-ci sera au centre de cette étude. En effet, les relations de simulation préservent les propriétés de sûreté [AL91] mais plus généralement elles préservent l'ensemble des propriétés exprimant une quantification universelle sur les chemins comme LTL [Pnu77] ou ACTL [BCG88].

La sémantique **AASAP** peut être présentée comme une relaxation de la sémantique classique **ASAP**, pour « as soon as possible », *i.e.* dès que possible. Un système muni d'une sémantique **ASAP** doit réagir instantanément, dès que possible à toutes ses entrées. Dans la sémantique **AASAP**, un paramètre est passé, le rationnel  $\Delta$ , qui permet de quantifier le temps de réaction du système. Les

caractéristiques principales de cette sémantique peuvent donc être résumées ainsi :

- Une transition franchissable ne doit pas nécessairement être franchie instantanément. Elle peut l'être plus tard, mais au plus après  $\Delta$  unités de temps.
- Le temps d'envoi et de réception des messages est pris en compte : une distinction est faite entre l'envoi d'un message par un automate et sa réception par un autre. Cependant, le délai qui sépare ces deux actions est bornée par  $\Delta$ .
- Enfin, la précision des horloges est relâchée. Plus précisément, les gardes sont élargies du paramètre  $\Delta$ .

La sémantique AASAP de  $\mathcal{A}$  pour le paramètre  $\Delta$  est notée  $\llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}}$ , et vérifie la propriété suivante.

**Proposition 8.2** (« Plus c'est rapide, mieux c'est » [DDR05a, Théorème 3]). *Soit  $\mathcal{A}$  un automate temporisé et  $\Delta_1, \Delta_2 \in \mathbb{Q}_{>0}$  tels que  $\Delta_2 \leq \Delta_1$ , alors le système de transitions  $\llbracket \mathcal{A} \rrbracket_{\Delta_2}^{\text{AASAP}}$  est simulé par  $\llbracket \mathcal{A} \rrbracket_{\Delta_1}^{\text{AASAP}}$ , ce qui est noté  $\llbracket \mathcal{A} \rrbracket_{\Delta_2}^{\text{AASAP}} \sqsubseteq \llbracket \mathcal{A} \rrbracket_{\Delta_1}^{\text{AASAP}}$ .*

Intuitivement, ceci signifie qu'augmenter la précision du système réduit le nombre de comportements admissibles. Ainsi, pour la vérification de propriétés universelles sur les chemins préservées par la simulation, si le modèle  $\llbracket \mathcal{A} \rrbracket_{\Delta_1}^{\text{AASAP}}$  vérifie la propriété, alors le modèle  $\llbracket \mathcal{A} \rrbracket_{\Delta_2}^{\text{AASAP}}$  la vérifie également.

Afin de faire le lien avec un système réel implémentant l'automate temporisé, une autre sémantique est introduite, représentant le comportement d'une plate-forme d'exécution réelle simulant l'automate temporisé. Nous décrivons le modèle considéré pour la plate-forme d'exécution. La procédure simulant le comportement de l'automate temporisé exécute de façon répétée le cycle d'instructions suivant :

1. La date courante est lue dans le registre d'horloge du processeur et stockée dans une variable notée  $T$ .
2. La liste des signaux entrants est mise à jour : les senseurs sont testés pour détecter si de nouveaux signaux entrants ont été envoyés par les autres composantes du système.
3. Les contraintes d'horloges des transitions sortantes de l'état courant du système sont évaluées avec la valeur  $T$ . Si une de ces contraintes est satisfaite, alors une des transitions franchissables est tirée.
4. Le tour suivant est activé.

La sémantique de programme de  $\mathcal{A}$  dépend de deux paramètres  $\Delta_P$  et  $\Delta_L$  et est notée  $\llbracket \mathcal{A} \rrbracket_{\Delta_L, \Delta_P}^{\text{Prog}}$ . Ces deux paramètres représentent la performance de la plate-forme et celle-ci doit vérifier les deux propriétés suivantes :

- (i) Le registre d'horloges est incrémenté toutes les  $\Delta_P$  unités de temps.
- (ii) Le temps nécessaire pour réaliser un cycle d'instructions est borné par la valeur  $\Delta_L$ .

Le résultat fondamental établissant le lien entre les deux sémantiques introduites précédemment est le suivant :

**Théorème 8.3** (Simulation [DDR05a, Théorème 4]). *Soit  $\mathcal{A}$  un automate temporisé et  $\Delta, \Delta_P, \Delta_L \in \mathbb{Q}_{>0}$  trois paramètres rationnels. Alors si ces trois paramètres vérifient l'inégalité  $3\Delta_L + 4\Delta_P \leq \Delta$ , les systèmes de transitions temporisés associés vérifient :*

$$\llbracket \mathcal{A} \rrbracket_{\Delta_L, \Delta_P}^{\text{Prog}} \sqsubseteq \llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}}$$

Ce résultat établit que si la plate-forme d'exécution est assez rapide et assez précise, alors l'ensemble de ses comportements est inclus dans la sémantique AASAP. La sémantique AASAP est donc un raffinement du comportement d'un système réel implémentant l'automate temporisé.

Nous présentons enfin un dernier résultat, établissant que la sémantique AASAP peut être simulée par une transformation syntaxique des automates.

**Théorème 8.4** ([DDR05a, Théorème 8]). *Soit  $\mathcal{A}$  un automate temporisé et  $\Delta \in \mathbb{Q}_{>0}$  un paramètre rationnel. Il est possible de construire de façon effective un automate temporisé  $\mathcal{F}(\mathcal{A}, \Delta)$  vérifiant :*

$$\llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}} \subseteq \llbracket \mathcal{F}(\mathcal{A}, \Delta) \rrbracket \text{ et } \llbracket \mathcal{F}(\mathcal{A}, \Delta) \rrbracket \subseteq \llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}}$$

L'opération principale mise en œuvre dans la définition de  $\mathcal{F}(\mathcal{A}, \Delta)$  est l'élargissement des gardes et des invariants de la valeur  $\Delta$ . Ceci consiste à remplacer toute contrainte d'horloges  $g$  par la contrainte d'horloges  $g_{\Delta}$  définie par induction sur  $g$  par :

$$g_{\Delta} = \begin{cases} \text{tt} & \text{si } g = \text{tt} \\ x \prec c + \Delta & \text{si } g = x \prec c \text{ avec } \prec \in \{<, \leq\} \\ c - \Delta \prec x & \text{si } g = c \prec x \text{ avec } \prec \in \{<, \leq\} \\ (g_1)_{\Delta} \wedge (g_2)_{\Delta} & \text{si } g = g_1 \wedge g_2 \\ (g_1)_{\Delta} \vee (g_2)_{\Delta} & \text{si } g = g_1 \vee g_2 \end{cases}$$

Dans la suite, pour simplifier les preuves, nous considérerons que la transformation  $\mathcal{F}(\mathcal{A}, \Delta)$  consiste uniquement en ces élargissements et nous noterons  $\mathcal{A}_{\Delta}$  l'automate obtenu. De plus, nous supposons que les gardes des automates temporisés considérés ne contiennent que des inégalités larges. Ceci n'est pas restrictif puisque l'opération d'élargissement rend larges les inégalités strictes. Cette opération d'élargissement est l'aspect le plus important de la sémantique AASAP, c'est pourquoi la simplification que nous adoptons est justifiée.

### 8.2.2 Vérification « robuste »

Nous présentons à présent une notion de vérification robuste inspirée de la sémantique AASAP.

**Définition 8.5** (Propriété de chemins). *Une propriété de chemins est une application  $\mathcal{P}$  de l'ensemble des chemins dans  $\{\text{tt}, \text{ff}\}$ . Étant donné un chemin  $\pi$ , nous écrirons  $\pi \models \mathcal{P}$  si et seulement si  $\mathcal{P}(\pi) = \text{tt}$ .*

**Définition 8.6** (Satisfaction robuste). *Étant donné une propriété de chemins  $\mathcal{P}$  et un automate temporisé  $\mathcal{A}$  d'état initial  $(\ell_0, v_0)$ , nous définissons la relation de satisfaction robuste  $\models$  ainsi :*

$$\mathcal{A} \models \mathcal{P} \stackrel{\text{def}}{\iff} \exists \Delta > 0 \text{ tel que pour tout chemin } \pi \text{ de } \llbracket \mathcal{A}_{\Delta} \rrbracket \text{ issu de } (\ell_0, v_0), \pi \models \mathcal{P}.$$

Cette relation représente donc le fait qu'il est possible d'introduire un certain élargissement  $\Delta > 0$  de l'automate  $\mathcal{A}$  qui assure que tous les comportements de  $\mathcal{A}_{\Delta}$  vérifient la propriété  $\mathcal{P}$ . Le problème de vérification associé est le suivant.

**Définition 8.7** (Vérification robuste). *Étant donné une propriété de chemins  $\mathcal{P}$  et un automate temporisé  $\mathcal{A}$ , le problème de la vérification robuste de  $\mathcal{P}$  par  $\mathcal{A}$  consiste à décider si  $\mathcal{A} \models \mathcal{P}$ .*

Dans la suite de ce chapitre, nous nous intéresserons toujours à ce problème, pour différentes classes de propriétés de chemin. Le théorème suivant exprime le lien obtenu grâce aux résultats de la sous-section précédente entre la vérification robuste et l'implémentabilité.

**Théorème 8.8** (La robustesse implique l'implémentabilité). *Soit  $\mathcal{A}$  un automate temporisé et  $\mathcal{P}$  une propriété de chemins préservée par la relation de simulation existant entre  $\llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}}$  et  $\llbracket \mathcal{A} \rrbracket^{\text{Prog}}$ . Si  $\mathcal{A} \equiv \mathcal{P}$  alors il existe une plate-forme d'exécution implémentant l'automate  $\mathcal{A}$  et dont tous les comportements vérifient la propriété  $\mathcal{P}$ .*

**Preuve.** Le résultat découle facilement des théorèmes 8.4 et 8.3. En effet, supposons que  $\mathcal{A} \equiv \mathcal{P}$ . Alors il existe un certain  $\Delta \in \mathbb{Q}_{>0}$  tel que tous les chemins de  $\llbracket \mathcal{A}_{\Delta} \rrbracket$  issus de  $(\ell_0, v_0)$  vérifient la propriété  $\mathcal{P}$ . D'après le théorème 8.4,  $\llbracket \mathcal{A}_{\Delta} \rrbracket$  simule  $\llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}}$ . Choisissons alors  $\Delta_L, \Delta_P \in \mathbb{Q}_{>0}$  vérifiant  $3\Delta_L + 4\Delta_P \leq \Delta$ . D'après le théorème 8.3 de simulation, nous obtenons que  $\llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}}$  simule  $\llbracket \mathcal{A} \rrbracket^{\text{Prog}}$ . Par transitivité de la notion de simulation, nous obtenons que  $\llbracket \mathcal{A}_{\Delta} \rrbracket$  simule le système de transitions temporisé  $\llbracket \mathcal{A} \rrbracket^{\text{Prog}}$  représentant les comportements de l'implémentation. Comme la relation de simulation préserve la propriété  $\mathcal{P}$ , ceci implique que l'ensemble des comportements de  $\llbracket \mathcal{A} \rrbracket^{\text{Prog}}$  vérifient la propriété  $\mathcal{P}$ , ce qu'il fallait démontrer.  $\square$

Dans la suite, nous allons présenter des algorithmes permettant de résoudre le problème de la vérification robuste pour des propriétés d'accessibilité, des propriétés d'accessibilité répétée, des propriétés exprimées dans la logique LTL et enfin pour certaines propriétés quantitatives exprimées dans la logique MTL.

**Propriétés d'accessibilité.** Nous présentons ici les résultats de [Pur98, DDMR04a] dans lesquels le problème de la vérification robuste de propriétés d'accessibilité est étudié. Certaines hypothèses sont faites sur les automates temporisés, que nous reprenons ici.

Un *cycle de progrès* dans l'automate des régions de  $\mathcal{A}$  est un cycle le long duquel toutes les horloges sont remises à zéro et qui n'est pas réduit à la région initiale (*i.e.* la région contenant la valuation  $\mathbf{0}$ ).

**Restrictions.** Nous restreignons notre étude aux automates temporisés  $\mathcal{A}$  vérifiant les conditions suivantes :

- les horloges sont bornées par une constante  $M$ ,
- tous les cycles dans l'automate des régions  $\mathcal{R}(\mathcal{A})$  sont des cycles de progrès.

La première hypothèse, déjà discutée dans la sous-section 7.5.2 page 213, n'est pas restrictive puisque tout automate temporisé peut être transformé afin de vérifier cette propriété. Remarquons que ceci implique que tout chemin de temps divergent contient un nombre infini d'actions discrètes. Dans la suite de ce chapitre, nous ne considérerons que des chemins de temps divergent. Le second point est une restriction classique [Pur98] éliminant les mots Zeno. Comme cela est mentionné dans [DDMR04a], celle-ci est moins restrictive que les hypothèses classiques imposant aux chemins d'être fortement non Zeno (voir par exemple [AMPS98]).

Sous ces hypothèses, le problème de la vérification de propriétés de sûreté exprimées comme la non accessibilité de certains états cibles est décidable. Afin de présenter ce résultat, nous introduisons quelques notations. Rappelons que, pour un automate temporisé  $\mathcal{A}$ ,  $\text{Acc}(\mathcal{A})$  représente l'ensemble des configurations accessibles dans l'automate temporisé  $\mathcal{A}$  (voir définition page 37). Nous définissons l'ensemble suivant :

$$\text{Acc}^*(\mathcal{A}) = \bigcap_{\Delta > 0} \text{Acc}(\mathcal{A}_{\Delta})$$

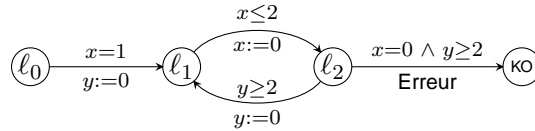
**Théorème 8.9** ([Pur98, DDMR04a]). *Soit  $\mathcal{A}$  un automate temporisé d'état initial  $(\ell_0, v_0)$  et  $\text{KO}$  un ensemble d'états de contrôle de  $\mathcal{A}$ . Nous avons alors :*

**Algorithme 12** Calcul de l'ensemble  $\text{Acc}^*(\mathcal{A})$ **Entrée :**  $\mathcal{A}$  un automate temporisé.**Sortie :** L'ensemble de configurations  $\text{Acc}^*(\mathcal{A})$ .

- 1: Calculer l'automate des régions  $\mathcal{R}(\mathcal{A}) = (\Gamma, \gamma_0, \rightarrow)$ .
- 2: Calculer l'ensemble  $S$  des composantes fortement connexes de  $\mathcal{R}(\mathcal{A})$ .
- 3:  $J^* := \{\gamma_0\}$ ;
- 4:  $J^* := \text{Acc}(\mathcal{R}(\mathcal{A}), J^*)$ ;<sup>1</sup>
- 5: **Tant Que**  $\exists S \in \mathcal{S}$  tel que  $S \not\subseteq J^*$  et  $J^* \cap S \neq \emptyset$  **Faire**
- 6:    $J^* := J^* \cup S$ ;
- 7:    $J^* := \text{Acc}(\mathcal{R}(\mathcal{A}), J^*)$ ;
- 8: **Fin Tant Que**
- 9: **Retourner**  $J^*$ ;

1.  $\exists \Delta > 0$  tel que  $\text{Acc}(\mathcal{A}_\Delta) \cap \text{KO} = \emptyset$  est équivalent à  $\text{Acc}^*(\mathcal{A}) \cap \text{KO} = \emptyset$ ,
2. tester si  $\text{Acc}^*(\mathcal{A}) \cap \text{KO} = \emptyset$  est décidable et PSPACE-complet.

Le premier point de ce théorème est un résultat de nature topologique. Le second point repose sur la construction du graphe des régions. L'algorithme 12 permet de résoudre ce point. Celui-ci consiste à calculer les composantes fortement connexes du graphe des régions, puis à les ajouter à l'ensemble des états accessibles si elles sont en partie accessibles. En effet, si une composante fortement connexe est en partie accessible, alors elle l'est entièrement. Plus précisément, dans une composante fortement connexe, l'élargissement des gardes permet de joindre entre eux n'importe quels éléments de cette composante en accumulant les déviations. Réciproquement, ceci constitue le seul moyen pour une configuration d'appartenir à  $\text{Acc}^*(\mathcal{A})$  mais pas à  $\text{Acc}(\mathcal{A})$ .

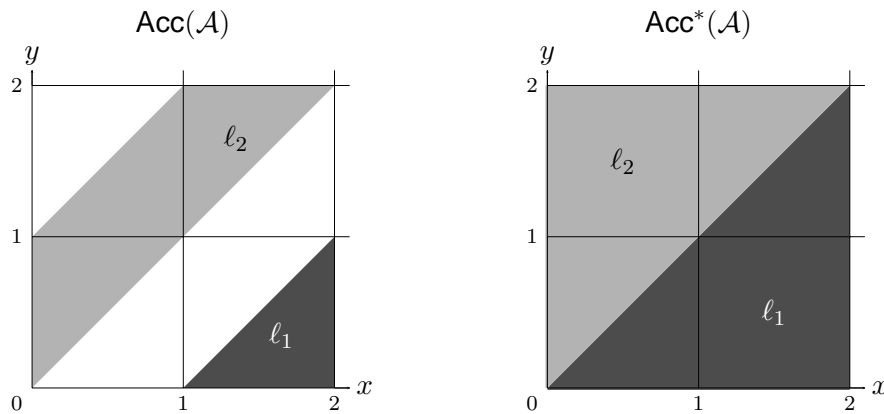
FIG. 8.2 – Un automate temporisé  $\mathcal{A}$ .

**Exemple 8.10** ([Pur98, DDMR04a]). Considérons l'automate temporisé représenté sur la figure 8.2. Nous pouvons calculer pour cet automate temporisé l'ensemble  $\text{Acc}(\mathcal{A})$  des configurations accessibles dans la sémantique classique et l'ensemble  $\text{Acc}^*(\mathcal{A})$  défini précédemment comme l'ensemble des configurations accessibles dans  $\mathcal{A}_\Delta$  pour tout paramètre  $\Delta$  strictement positif. Nous obtenons, pour les états de contrôle  $l_1$  et  $l_2$ , les deux ensembles représentés sur la figure 8.3. La différence entre ces deux ensembles provient du cycle existant entre  $l_1$  et  $l_2$ .  $\lrcorner$

**Remarque 8.11.** Notons que la sémantique de la satisfaction robuste n'est pas classique. En effet, il peut arriver qu'un modèle  $\mathcal{A}$  ne vérifie pas robustement la propriété  $\mathcal{P}$  et qu'il ne vérifie pas non plus robustement la propriété  $\neg \mathcal{P}$ . Ainsi, la proposition  $\mathcal{A} \not\models \mathcal{P}$  n'est pas équivalente à la proposition  $\mathcal{A} \models \neg \mathcal{P}$ . Par exemple, considérons l'automate  $\mathcal{A}$  de l'exemple 8.10 et la propriété de sûreté  $\mathcal{P} = \mathbf{G}(\neg \text{KO})$ . La négation est  $\neg \mathcal{P} = \mathbf{F}(\text{KO})$ . Nous pouvons observer que dans  $\llbracket \mathcal{A} \rrbracket$ , la transition Erreur

<sup>1</sup> $\text{Acc}(\mathcal{R}(\mathcal{A}), J^*)$  désigne l'ensemble des états accessibles depuis  $J^*$  dans  $\mathcal{R}(\mathcal{A})$ .



FIG. 8.3 – Différences entre  $\text{Acc}(\mathcal{A})$  et  $\text{Acc}^*(\mathcal{A})$ .

n'est jamais franchissable, nous avons donc  $\mathcal{A} \not\models \neg\mathcal{P}$  ce qui implique  $\mathcal{A} \not\models \mathcal{P}$ . Au contraire, dans  $\llbracket \mathcal{A}_\Delta \rrbracket$ , pour tout  $\Delta > 0$ , cette transition est franchissable (d'après la figure 8.3). Ce dernier point entraîne  $\mathcal{A} \not\models \mathcal{P}$ .  $\lrcorner$

Récemment, une version symbolique de cet algorithme a été proposée dans [DK06]. Ce nouvel algorithme est une adaptation de l'algorithme d'analyse en avant à la volée sur les zones (algorithme 2 page 155). Essentiellement, la seule différence consiste à ajouter à l'ensemble des zones accessibles des cycles de zones dès lors qu'une des zones de ce cycle est partiellement atteinte, comme cela est fait avec les régions dans l'algorithme 12.

Dans les sections suivantes, nous proposons des algorithmes pour la vérification robuste de propriétés plus générales que les propriétés de sûreté. Nous prouvons ainsi que les problèmes de la vérification robuste de propriétés co-Büchi, de LTL et de réponses en temps borné sont décidables.

### 8.3 Vérification robuste de conditions co-Büchi

Dans cette section, nous nous intéressons à des conditions « co-Büchi » : étant donné un ensemble  $B$  d'états de contrôle de l'automate, un chemin  $\pi$  vérifie la condition  $\text{co-Büchi}(B)$  si et seulement si sa trace contient un nombre fini de transitions entrant dans un état de  $B$ . D'après les définitions 8.6 et 8.7, ceci définit les notions de satisfaction robuste et de vérification robuste d'une condition co-Büchi dans un automate temporisé. Nous étendons également la notion de satisfaction d'une condition co-Büchi à l'automate des régions : celui-ci satisfait une condition de co-Büchi  $B$  ne portant que sur les états de contrôle si et seulement si tout chemin issu de l'état initial  $\gamma_0$  traverse les états de  $B$  un nombre fini de fois.

Nous adaptons la construction présentée précédemment dans l'algorithme 12 en construisant une extension du graphe des régions prenant en compte les possibles « déviations » introduites dans l'automate temporisé sous-jacent lorsque la satisfaction robuste est prise en compte.

**Définition 8.12** (Automate des régions étendu). *Soit  $\mathcal{A}$  un automate temporisé et  $\mathcal{R}(\mathcal{A})$  son automate des régions. Nous définissons l'automate des régions étendu de  $\mathcal{A}$ , noté  $\mathcal{R}^*(\mathcal{A})$ , comme suit :*

- les états de  $\mathcal{R}^*(\mathcal{A})$  sont les états de  $\mathcal{R}(\mathcal{A})$ , i.e. les paires  $(\ell, r)$  où  $\ell$  est un état de contrôle de  $\mathcal{A}$  et  $r$  est une région de l'automate  $\mathcal{A}$ ,

- les transitions de  $\mathcal{R}^*(\mathcal{A})$  sont les transitions de  $\mathcal{R}(\mathcal{A})$  (nous supposons que les étiquettes des transitions sont les noms des transitions dans  $\mathcal{A}$ ), ainsi que les transitions  $(\ell, r) \xrightarrow{\gamma} (\ell', r')$  où  $\bar{r} \cap \bar{r}' \neq \emptyset$  et  $(\ell, r')$  appartient à une composante fortement connexe de  $\mathcal{R}(\mathcal{A})$ .

Nous introduisons une notion de distance entre les chemins pour comparer les trajectoires dans les automates temporisés.

**Définition 8.13** (Distance). Soit  $\mathcal{A}$  un automate temporisé. Notons  $\|\cdot\|_\infty$  la norme infinie classique. La distance entre deux configurations  $(\ell, v)$  et  $(\ell', v')$  de  $\mathcal{A}$ , notée  $d((\ell, v), (\ell', v'))$ , est définie par  $d((\ell, v), (\ell', v')) = \|v' - v\|_\infty$  si  $\ell = \ell'$  et par  $+\infty$  sinon.

La distance entre deux chemins  $\pi = (\pi_i)$  et  $\pi' = (\pi'_i)$  de même longueur est définie naturellement comme le maximum des distances  $d(\pi_i, \pi'_i)$  entre les configurations intermédiaires.

### Décidabilité de la vérification robuste de conditions co-Büchi.

**Théorème 8.14.** Soit  $\mathcal{A}$  un automate temporisé et  $B$  un ensemble d'états de contrôle de  $\mathcal{A}$ . Alors

$$\mathcal{A} \models \text{co-Büchi}(B) \iff \mathcal{R}^*(\mathcal{A}) \models \text{co-Büchi}(B)$$

La preuve de ce résultat repose sur les deux lemmes suivants :

**Lemme 8.15** ([DDMR04b, Théorème 27]). Soit  $\mathcal{A}$  un automate temporisé ayant  $n$  horloges, soit  $\alpha < \frac{1}{n}$  et soit  $k$  un entier naturel. Il existe un certain  $\Delta > 0$  tel que pour tout chemin  $\pi$  dans  $\llbracket \mathcal{A}_\Delta \rrbracket$  le long d'une trace  $T$  de longueur inférieure ou égale à  $k$ , il existe un chemin  $\pi'$  dans  $\llbracket \mathcal{A} \rrbracket$  le long de la même trace  $T$  tel que la distance entre les deux chemins  $\pi$  et  $\pi'$  est strictement inférieure à  $\alpha$ .

**Lemme 8.16** ([DDMR04b, Theorem 25]). Soit  $\mathcal{A}$  un automate temporisé, soit  $\tau$  la trace d'un chemin cyclique de  $\mathcal{R}(\mathcal{A})$  et soit  $(\ell, r)$  un état de  $\tau$ . Alors pour toutes valuations  $v$  et  $v'$  appartenant à  $\bar{r}$ , pour tout  $\Delta > 0$ , il existe un chemin dans  $\llbracket \mathcal{A}_\Delta \rrbracket$  le long de la trace  $\tau^k$  pour un certain  $k \in \mathbb{N}$ , issu de  $(\ell, v)$  et finissant dans  $(\ell, v')$ .

L'intuition du premier lemme est que si nous donnons une borne sur la longueur des chemins, alors il est possible de choisir  $\Delta > 0$  suffisamment petit pour assurer que l'ensemble des chemins de  $\llbracket \mathcal{A}_\Delta \rrbracket$  puissent être approchés par des chemins de  $\llbracket \mathcal{A} \rrbracket$ . L'intuition du second lemme est que deux états de  $\mathcal{A}$  appartenant à un certain chemin cyclique de l'automate des régions  $\mathcal{R}(\mathcal{A})$  peuvent être connectés (en utilisant un chemin de  $\llbracket \mathcal{A}_\Delta \rrbracket$ ) simplement en itérant le cycle de  $\mathcal{R}(\mathcal{A})$ . Naturellement, le nombre de fois que ce cycle doit être pris dépend du choix de  $\Delta$ , plus  $\Delta$  est petit, plus ce nombre de tours doit être grand.

**Preuve du théorème 8.14.** Nous supposons d'abord que  $\mathcal{A} \models \text{co-Büchi}(B)$  et nous choisissons un certain  $\Delta > 0$  tel que tout chemin de  $\llbracket \mathcal{A}_\Delta \rrbracket$  issu de  $(\ell_0, v_0)$  vérifie la condition de co-Büchi condition.

Raisonnons par l'absurde et supposons donc que  $\mathcal{R}^*(\mathcal{A}) \not\models \text{co-Büchi}(B)$ . Choisissons alors un chemin  $\pi$  ne vérifiant pas la condition de co-Büchi. Celui-ci passe donc un nombre infini de fois par un état de contrôle de  $B$ . Puisque  $B$  est fini, il existe alors un état de contrôle  $f \in B$  dans lequel le chemin entre un nombre infini de fois. Nous divisons alors le chemin  $\pi$  en une séquence de sous-chemins ne contenant aucune transition étiquetée par  $\gamma$  :  $\pi = \pi_0 \xrightarrow{\gamma} \pi_1 \xrightarrow{\gamma} \dots$ . Alors chaque  $\pi_i$  est un chemin fini non vide (excepté le dernier qui peut être infini) dans  $\mathcal{R}(\mathcal{A})$  le long d'une certaine trace  $T_i$ . Pour tout indice  $i$ , notons  $r_i$  la première région de  $\pi_i$  et  $r'_i$  la dernière. Par construction de  $\mathcal{R}^*(\mathcal{A})$ , nous obtenons alors  $\bar{r}'_i \cap \bar{r}_{i+1} \neq \emptyset$ . Nous utilisons alors le lemme suivant pour construire un nouveau chemin dans  $\llbracket \mathcal{A}_\Delta \rrbracket$  issu de l'état initial  $(\ell_0, u_0)$  de  $\mathcal{A}$  et ne vérifiant pas la condition de co-Büchi.

**Lemme 8.17.** Soit  $\pi$  un chemin de  $\mathcal{R}(\mathcal{A})$  étiqueté par  $T$ , issu de  $(\ell, r)$  et finissant dans  $(\ell', r')$  tel qu'il existe une transition  $(\ell', r') \xrightarrow{\gamma} (\ell', r'')$  dans  $\mathcal{R}^*(\mathcal{A})$  (due à un cycle le long d'une trace  $\tau$ ). Alors, pour tout  $\Delta > 0$ ,

1. pour chaque valuation  $v' \in \overline{r'}$ , il existe une valuation  $v \in \overline{r}$  et un chemin dans  $\llbracket \mathcal{A}_\Delta \rrbracket$  entre  $(\ell, v)$  et  $(\ell', v')$  le long de la trace  $T$ ;
2. pour chaque valuation  $v' \in \overline{r'} \cap \overline{r''}$  et pour chaque valuation  $v'' \in \overline{r''}$ , il existe un chemin dans  $\llbracket \mathcal{A}_\Delta \rrbracket$  le long de la trace  $\tau^k$  (pour un certain  $k \geq 0$ ) entre  $(\ell', v')$  et  $(\ell', v'')$ .

**Preuve du lemme 8.17.** Étant donné un automate temporisé  $\mathcal{A}$ , un ensemble de configurations  $Q'$  de  $\mathcal{A}$  et une trace  $T$  de  $\mathcal{A}$ , nous notons  $\text{Acc}^T(\mathcal{A}, Q')$  l'ensemble des configurations accessibles dans  $\mathcal{A}$  depuis  $Q'$  le long de  $T$ . Il est facile de vérifier que l'ensemble  $\bigcap_{\Delta > 0} \text{Acc}^T(\mathcal{A}_\Delta, (\ell, \overline{r}))$  est fermé et contient la région  $r'$  par hypothèse. Il contient donc la fermeture  $\overline{r'}$  de  $r'$ . Ainsi, pour tout  $\Delta > 0$ ,  $(\ell', \overline{r'}) \subseteq \text{Acc}^T(\mathcal{A}_\Delta, (\ell, \overline{r}))$ , ce qui conclut le premier point du lemme.

Le second point est une conséquence directe du lemme 8.16 : le cycle le long de la trace  $\tau$  peut être itéré et permet ainsi de relier entre eux n'importe quels états de  $r''$ .  $\square$

Appliquons le premier point de ce lemme au chemin  $\pi_i$  pour tout indice  $i$  pour lequel ce chemin est fini. Pour toute valuation  $v'_i \in \overline{r'_i} \cap \overline{r_{i+1}}$ , il existe une valuation  $v_i \in \overline{r_i}$  et un chemin réel dans  $\llbracket \mathcal{A}_\Delta \rrbracket$  le long de la trace  $T_i$  depuis la configuration  $(\ell_i, v_i)$  et atteignant  $(\ell'_i, v'_i)$ . Afin de rassembler les différents sous-chemins, nous appliquons le deuxième point du lemme. Pour chaque indice  $i$ , nous obtenons ainsi un chemin réel de  $\llbracket \mathcal{A}_\Delta \rrbracket$  le long de la trace  $\tau_i^{k_i}$  pour un certain  $k_i \geq 0$ . Ce chemin est issu de la configuration  $(\ell'_i, v'_i)$  et atteint la configuration  $(\ell_{i+1}, v_{i+1})$ . Notons que pour le premier sous-chemin  $\pi_0$ , puisque la région initiale  $r_0$  est égale à  $\{u_0\}$ , nous obtenons  $v_0 = u_0$  et donc le chemin réel correspondant est issu de la configuration  $(\ell_0, u_0)$ .

Pour conclure, nous distinguons deux cas : si le chemin  $\pi$  contient un nombre infini de transitions  $\gamma$  alors il suffit de réunir tous les sous-chemins. Nous obtenons en effet un chemin réel de  $\llbracket \mathcal{A}_\Delta \rrbracket$  le long de la trace  $T_0 \tau_1^{k_1} T_1 \dots \tau_i^{k_i} T_i \dots$  qui passe infiniment souvent par l'état de contrôle  $f$ .

Si au contraire le nombre d'occurrences de transitions étiquetées par  $\gamma$  est fini, alors le chemin  $\pi_p$  est un chemin infini de  $\mathcal{R}(\mathcal{A})$ . Nous pouvons donc, d'après la propriété de bisimulation de temps abstrait de l'automate des régions, fixer une valuation  $v_p \in r_p$  et construire un chemin réel de  $\llbracket \mathcal{A} \rrbracket$  (et donc aussi de  $\llbracket \mathcal{A}_\Delta \rrbracket$ ) issu de la configuration  $(\ell_p, v_p)$  et suivant le chemin  $\pi_p$ . Concaténant ce chemin aux premiers sous-chemins obtenus plus haut, nous obtenons un chemin réel de  $\llbracket \mathcal{A}_\Delta \rrbracket$  le long de la trace  $T_0 \tau_1^{k_1} T_1 \dots T_{p-1} \tau_p^{k_p} T_p$ . Ce chemin passe infiniment souvent par l'état de contrôle  $f$  puisque c'est le cas de  $T_p$ .

Dans les deux cas, nous obtenons donc une contradiction avec l'hypothèse initiale.

Réciproquement, supposons que  $\mathcal{A} \not\models \text{co-Büchi}(B)$ . Ceci implique que pour tout  $\Delta > 0$ , il existe un chemin  $\pi_\Delta$  dans  $\llbracket \mathcal{A}_\Delta \rrbracket$  passant un nombre infini de fois par un état de contrôle de  $B$ . Puisque  $B$  est fini, il existe un état de contrôle  $f \in B$  tel que, pour tout  $\Delta > 0$ , il existe  $0 < \Delta' < \Delta$  tel que le chemin  $\pi_{\Delta'}$  de  $\llbracket \mathcal{A}_{\Delta'} \rrbracket$  passe infiniment souvent par  $f$ . Nous fixons un tel  $f$ .

Afin d'appliquer le lemme 8.15, nous choisissons un certain  $\alpha < 1/n$  et posons  $k = 2W + 1$  où  $W$  est le nombre d'états de l'automate  $\mathcal{R}(\mathcal{A})$ . Le lemme 8.15 entraîne alors l'existence d'un certain  $\Delta > 0$  pour lequel tout chemin de  $\llbracket \mathcal{A}_\Delta \rrbracket$  de longueur inférieure ou égale à  $k$  peut être approché par un chemin de  $\llbracket \mathcal{A} \rrbracket$  le long de la même trace et à une distance inférieure à  $\frac{\alpha}{2}$ . Nous fixons un tel  $\Delta > 0$  et choisissons un paramètre  $0 < \Delta' < \Delta$  tel que le chemin associé  $\pi_{\Delta'}$  vérifie la condition de Büchi  $f$ , i.e. passe infiniment souvent par  $\{f\}$ . Nous notons simplement  $\pi$  ce chemin.

Nous montrons à présent que, pour tout chemin (fini) de  $\llbracket \mathcal{A}_{\Delta'} \rrbracket$ , nous pouvons construire un chemin dans  $\mathcal{R}^*(\mathcal{A})$  suivant les mêmes transitions, hormis le fait qu'il peut franchir des transitions  $\gamma$  supplémentaires. Ceci est suffisant pour obtenir la réciproque. En effet, nous souhaitons montrer que  $\mathcal{R}^*(\mathcal{A}) \not\equiv \text{co-Büchi}(B)$ . Or, en choisissant un préfixe fini de  $\pi$  contenant  $k + 1$  occurrences de l'état  $f$ , nous obtenons un chemin dans  $\mathcal{R}^*(\mathcal{A})$  contenant  $k + 1$  occurrences de  $f$ . Ceci implique qu'il existe un cycle dans  $\mathcal{R}^*(\mathcal{A})$ , accessible depuis l'état initial. En particulier, il existe un chemin de  $\mathcal{R}^*(\mathcal{A})$  vérifiant la condition de Büchi  $\{f\}$ .

Soit  $\rho$  un chemin de  $\llbracket \mathcal{A}_{\Delta'} \rrbracket$ . Nous démontrons par induction sur la longueur de ce chemin que nous pouvons construire un chemin  $\nu$  dans  $\mathcal{R}^*(\mathcal{A})$  dont la trace, une fois les transitions  $\gamma$  retirées, est la même que celle de  $\rho$  et qui de plus reste à distance inférieure à  $\frac{\alpha}{2}$  de  $\rho$ . Dans la suite, pour simplifier les notations, nous écrirons simplement, étant données deux valuations  $u$  et  $v$ ,  $d(u, v)$  pour  $\|v - u\|_{\infty}$ . La distance entre une valuation  $u$  et une région  $r$ , notée  $d(u, r)$ , est définie comme la borne inférieure de l'ensemble des distances  $d(u, v)$  où  $v$  décrit l'ensemble des valuations de la région  $r$ .

- si la longueur de  $\rho$  est inférieure à  $k$ , alors nous pouvons appliquer directement le lemme 8.15 : il existe un chemin correspondant dans  $\llbracket \mathcal{A} \rrbracket$  qui suit les mêmes transitions que  $\pi$  et donc un chemin dans  $\mathcal{R}(\mathcal{A})$  (et a fortiori dans  $\mathcal{R}^*(\mathcal{A})$ ) suivant les mêmes transitions ;
- supposons que la longueur de  $\rho$  vaut  $N$  et que nous avons résolu le problème pour les chemins de longueur strictement inférieure à  $N$ . Nous posons les notations suivantes :

$$\rho = (\ell_0, v'_0) \xrightarrow{\sigma_1} (\ell_1, v'_1) - \dots - \xrightarrow{\sigma_{N-1}} (\ell_{N-1}, v'_{N-1}) \xrightarrow{\sigma_N} (\ell_N, v'_N).$$

D'après l'hypothèse d'induction, il existe un chemin  $\nu_{N-1}$  dans  $\mathcal{R}^*(\mathcal{A})$  tel que

$$\nu_{N-1} = (\ell_0, u_0) \xrightarrow{\gamma^* \sigma_1 \gamma^*} (\ell_1, u_1) - \dots - \xrightarrow{\gamma^* \sigma_{N-1} \gamma^*} (\ell_{N-1}, u_{N-1})$$

où les  $u_i$  sont des régions telles que, pour tout  $0 \leq i < N$ ,  $d(v'_i, u_i) < \frac{\alpha}{2}$ . D'après le lemme 8.15, nous pouvons construire un chemin  $\rho'$  dans  $\mathcal{A}$

$$\rho' = (\ell_{N-k}, v_{N-k}) \xrightarrow{\sigma_{N-k+1}} (\ell_{N-k+1}, v_{N-k+1}) - \dots - \xrightarrow{\sigma_N} (\ell_N, v_N)$$

tel que pour tout indice  $N - k \leq i \leq N$ ,  $d(v_i, v'_i) < \frac{\alpha}{2}$ . Nous considérons l'exécution correspondante dans  $\mathcal{R}(\mathcal{A})$  :

$$\nu' = (\ell_{N-k}, r_{N-k}) \xrightarrow{\sigma_{N-k+1}} (\ell_{N-k+1}, r_{N-k+1}) - \dots - \xrightarrow{\sigma_N} (\ell_N, r_N)$$

où pour tout  $i$ , nous posons  $r_i = [v_i]$ .

Pour chaque  $N - k \leq i \leq N$ , nous obtenons qu'il existe  $z_i \in u_i$  tel que  $d(z_i, v_i) < \alpha$ , ce qui implique que  $\overline{r_i} \cap \overline{u_i} \neq \emptyset$  (d'après [DDMR04b, Lemme 12]).

Comme la longueur de  $\nu'$  est égale à  $k$ , il existe  $N - k \leq p < q \leq N$  tel que  $(\ell_p, r_p) = (\ell_q, r_q)$  et donc  $(\ell_p, r_p)$  appartient à une composante fortement connexe de  $\mathcal{R}(\mathcal{A})$ . Comme  $\overline{r_p} \cap \overline{u_p} \neq \emptyset$ , il existe une transition  $(\ell_p, u_p) \xrightarrow{\gamma} (\ell_p, r_p)$  dans  $\mathcal{R}^*(\mathcal{A})$ . Ceci nous permet donc de concaténer  $\nu_{N-1}$  et  $\nu'$ . Le chemin obtenu est

$$\begin{aligned} \nu = (\ell_0, u_0) \xrightarrow{\gamma^* \sigma_1 \gamma^*} (\ell_1, u_1) - \dots - \xrightarrow{\gamma^* \sigma_p \gamma^*} (\ell_p, u_p) \xrightarrow{\gamma} (\ell_p, r_p) \xrightarrow{\sigma_{p+1}} (\ell_{p+1}, r_{p+1}) - \dots - \xrightarrow{\sigma_N} (\ell_N, r_N) \end{aligned}$$

et est un chemin de  $\mathcal{R}^*(\mathcal{A})$  vérifiant les conditions recherchées concernant sa trace et sa distance à  $\rho$ .

Ceci conclut donc la preuve de l'existence du chemin  $\nu$  et ainsi la preuve du théorème 8.14.  $\square$

Comme corollaire, nous obtenons :

**Corollaire 8.18.** *La vérification robuste des conditions co-Büchi pour des automates temporisés est un problème PSPACE-complet.*

**Preuve.** Nous procédons en deux temps, en montrant que le problème est à la fois dans la classe PSPACE, puisqu'il est complet pour cette classe.

- PSPACE-facile : L'automate des régions étendu  $\mathcal{R}^*(\mathcal{A})$  ne nécessite pas d'être construit pour être analysé. Ainsi, la condition co-Büchi peut être testée à la volée en testant sa condition complémentaire, *i.e.* en testant s'il existe un chemin infini dans  $\mathcal{R}^*(\mathcal{A})$  passant infiniment souvent par  $B$ . Ceci est équivalent à tester l'accessibilité d'un cycle de  $\mathcal{R}^*(\mathcal{A})$  contenant un état  $f \in B$ . Ceci constitue un problème classique de test d'accessibilité dans l'automate des régions qui appartient à la classe NLOGSPACE si l'automate des régions est donné. Il suffit en effet de deviner un chemin menant à une composante fortement connexe, puis deviner un chemin assurant qu'il s'agit d'une composante fortement connexe contenant un état de  $B$ . Notons qu'un état de  $\mathcal{R}^*(\mathcal{A})$  peut être stocké en espace polynomial et que le calcul des successeurs d'un état par les transitions de  $\mathcal{R}(\mathcal{A})$  est également réalisable en espace polynomial. Le cas des transitions  $\gamma$  diffère légèrement. Nous devons dans ce cas deviner le franchissement d'une telle transition et de la région cible, puis vérifier que la région cible est élément d'une composante fortement connexe ou, plus simplement, vérifier qu'il existe dans  $\mathcal{R}(\mathcal{A})$  un cycle autour de cette région. Finalement, cette procédure appartient donc à la classe NPSpace et, d'après le théorème de Savitch, à la classe PSPACE, comme annoncé.
- PSPACE-difficile : Nous faisons la preuve de ce point par réduction au problème de la vérification robuste de propriétés de sûreté (théorème 8.9). Considérons un automate temporisé  $\mathcal{A}$  avec un sous-ensemble  $B$  de ses états de contrôle. Définissons l'automate temporisé  $\mathcal{B}$  obtenu à partir de  $\mathcal{A}$  en ajoutant une boucle sur les états de contrôle appartenant à  $B$ . Alors décider la satisfaction robuste de la condition co-Büchi( $B$ ) pour cet automate  $\mathcal{B}$  est équivalent à décider la satisfaction robuste de la propriété de sûreté  $B$  pour  $\mathcal{A}$ , ce qui conclut la preuve.

Ceci conclut la preuve de ce corollaire.  $\square$

**Remarque 8.19.** Nous avons démontré le théorème 8.14 pour des conditions co-Büchi car ceci nous permet par la suite de décider les conditions exprimées en LTL. Cependant, les techniques mises en œuvre pour la démonstration de ce théorème peuvent être adaptées de sorte à traiter le cas de conditions de Büchi. Ceci nécessiterait de déplier une fois les composantes fortement connexes dans la construction de  $\mathcal{R}^*(\mathcal{A})$  afin d'obtenir une équivalence semblable à celle énoncée dans le théorème 8.14.  $\lrcorner$

## 8.4 Vérification robuste de LTL

Dans cette section, nous montrons comment les résultats précédents sur la vérification de conditions co-Büchi permettent de vérifier robustement des propriétés LTL sur les automates temporisés. Nous utilisons pour cela la construction classique d'automates de Büchi reconnaissant précisément les modèles des formules LTL et appliquons ensuite nos résultats pour les conditions co-Büchi.

**Définition 8.20** (Logique LTL). *La logique LTL sur l'ensemble fini d'actions  $\Sigma$  est définie par la grammaire suivante : (a décrit l'ensemble des actions  $\Sigma$ )*

$$\text{LTL} \ni \varphi ::= a \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2$$

**Sémantique de LTL.** Comme cela a été expliqué dans la section 8.2, nous devons définir la sémantique de LTL sur les chemins d'automate temporisés. À un chemin, correspond une unique trace et comme la satisfaction d'une formule de LTL pour un chemin ne dépend que de ses actions, celle-ci ne dépend que de la trace associée au chemin. Nous définissons donc la relation de satisfaction  $\models$  pour LTL sur les traces et disons qu'un chemin  $\pi$  satisfait une formule LTL si et seulement si sa trace  $\text{trace}(\pi)$  satisfait cette formule.

Nous supposons donc donnée une trace  $T = (\delta_i)_{i \in \mathbb{N}} \in \Sigma^\omega$ . Étant donné un entier naturel  $j \in \mathbb{N}$ , nous notons  $T^j$  la trace  $(\delta_i)_{i \geq j}$ . La relation de satisfaction pour LTL sur les traces est définie par induction par :

$$\begin{aligned} T \models a & \iff \delta_0 = a \\ T \models \varphi_1 \vee \varphi_2 & \iff T \models \varphi_1 \text{ ou } T \models \varphi_2 \\ T \models \neg\varphi & \iff T \not\models \varphi \\ T \models \mathbf{X}\varphi & \iff T^1 \models \varphi \\ T \models \varphi_1 \mathbf{U} \varphi_2 & \iff \exists i \geq 0 \text{ t.q. } T^i \models \varphi_2 \text{ et } \forall 0 \leq j < i, T^j \models \varphi_1 \end{aligned}$$

Dans la suite, nous écrirons simplement  $\pi \models \varphi$  pour signifier  $\text{trace}(\pi) \models \varphi$  et utiliserons les abréviations classiques comme  $\mathbf{F}\varphi$  (qui signifie  $\text{tt}\mathbf{U}\varphi$ ) ou  $\mathbf{G}\varphi$  (qui représente la formule  $\neg(\mathbf{F}(\neg\varphi))$ ).

**Remarque 8.21.** La sémantique que nous considérons ici est la sémantique « ponctuelle » dans laquelle les formules sont interprétées seulement lorsqu'une action a lieu, qui est sensiblement différente de la sémantique « continue » dans laquelle les actions peuvent être interprétées à tout instant (voir par exemple [Ras99, OW05] pour une discussion sur ces deux sémantiques).  $\lrcorner$

**Sémantique robuste de LTL.** La *relation de satisfaction robuste* pour LTL est dérivée de la définition générale 8.6 :

$$\mathcal{A} \models \varphi \iff \exists \Delta > 0 \text{ tel que } \forall \pi \text{ chemin de } \llbracket \mathcal{A}_\Delta \rrbracket \text{ issu de } (\ell_0, v_0), \pi \models \varphi.$$

Cette définition étend la définition proposée dans [DDMR04a] pour les propriétés de sûreté qui correspondent aux formules LTL de la forme  $\mathbf{G}(\neg a)$ .

**Vérification robuste de LTL.** Nous rappelons d'abord le résultat classique suivant liant la logique LTL aux automates de Büchi :

**Proposition 8.22** ([WVS83, Var96]). *Étant donnée une formule LTL  $\varphi$ , nous pouvons construire un automate de Büchi  $\mathcal{B}_\varphi$  (avec pour état initial  $q_\varphi$  et pour ensemble d'états répétés  $Q_\varphi$ ) qui accepte l'ensemble  $\{T \in \Sigma^\omega \mid T \models \varphi\}$ .*

Nous démontrons à présent que la vérification robuste de LTL est décidable.

**Théorème 8.23.** *Étant donné un automate temporisé  $\mathcal{A}$  et une formule de LTL  $\varphi$ , nous notons  $\mathcal{C} = \mathcal{A} \times \mathcal{B}_{\neg\varphi}$  l'automate temporisé (avec condition d'acceptation de Büchi  $Q_{\neg\varphi}$ ) obtenu en réalisant le produit synchronisé des deux automates  $\mathcal{A}$  et  $\mathcal{B}_{\neg\varphi}$  sur leurs actions. Nous obtenons alors l'équivalence suivante :*

$$\mathcal{A} \models \varphi \iff \mathcal{C} \models \text{co-Büchi}(L \times Q_{\neg\varphi}).$$

**Preuve.** Nous procédons par équivalence à partir de la négation de la première proposition :

$$\begin{aligned}
\mathcal{A} \not\models \varphi &\iff \forall \Delta > 0, \exists \pi \text{ chemin de } \llbracket \mathcal{A}_\Delta \rrbracket \text{ issu de } (\ell_0, v_0), \text{ tel que } \pi \not\models \varphi \\
&\iff \forall \Delta > 0, \exists \pi \text{ chemin de } \llbracket \mathcal{A}_\Delta \rrbracket \text{ issu de } (\ell_0, v_0), \text{ tel que } \pi \models \neg\varphi \\
&\iff \forall \Delta > 0, \exists \pi \text{ chemin de } \llbracket \mathcal{A}_\Delta \rrbracket \text{ issu de } (\ell_0, v_0), \text{ tel que la trace } T = \\
&\quad \text{trace}(\pi) \text{ associée à } \pi \text{ est acceptée par } \mathcal{B}_{\neg\varphi} \\
&\iff \forall \Delta > 0, \exists \pi \text{ chemin de } \llbracket \mathcal{A}_\Delta \rrbracket \text{ issu de } (\ell_0, v_0), \text{ tel que la trace } T = \\
&\quad \text{trace}(\pi) \text{ associée à } \pi \text{ est telle que } T \models \text{Büchi}(Q_{\neg\varphi}) \text{ dans } \mathcal{B}_{\neg\varphi} \\
&\quad (\text{par définition de } \mathcal{B}_{\neg\varphi} - \text{voir la proposition 8.22})
\end{aligned}$$

En notant  $\times$  la synchronisation sur les actions de  $\Sigma$ , il est facile de vérifier que les deux systèmes de transitions temporisés  $\llbracket \mathcal{A}_\Delta \rrbracket \times \mathcal{B}_{\neg\varphi}$  et  $\llbracket \mathcal{A} \times \mathcal{B}_{\neg\varphi} \rrbracket$  sont isomorphes. En particulier, l'ensemble des chemins qu'ils acceptent sont les mêmes. Nous obtenons donc :

$$\begin{aligned}
\mathcal{A} \not\models \varphi &\iff \forall \Delta > 0, \exists \pi \text{ chemin de } \llbracket \mathcal{A} \times \mathcal{B}_{\neg\varphi} \rrbracket \text{ issu de } ((\ell_0, v_0), q_{\neg\varphi}), \text{ tel que} \\
&\quad \text{la trace } T = \text{trace}(\pi) \text{ associée à } \pi \text{ est telle que } T \models \text{Büchi}(L \times Q_{\neg\varphi}) \\
&\quad (\text{où } L \text{ est l'ensemble des états de contrôle de } \mathcal{A}) \\
&\iff \forall \Delta > 0, \exists \pi \text{ chemin de } \llbracket \mathcal{C}_\Delta \rrbracket \text{ issu de } ((\ell_0, v_0), q_{\neg\varphi}), \text{ tel que} \\
&\quad \pi \models \text{Büchi}(L \times Q_{\neg\varphi})
\end{aligned}$$

Prenant à nouveau la négation, nous obtenons :

$$\begin{aligned}
\mathcal{A} \models \varphi &\iff \exists \Delta > 0, \text{ tel que } \forall \pi \text{ chemin de } \llbracket \mathcal{C}_\Delta \rrbracket \text{ issu de } ((\ell_0, v_0), q_{\neg\varphi}), \\
&\quad \pi \not\models \text{Büchi}(L \times Q_{\neg\varphi}) \\
&\iff \exists \Delta > 0, \text{ tel que } \forall \pi \text{ chemin de } \llbracket \mathcal{C}_\Delta \rrbracket \text{ issu de } ((\ell_0, v_0), q_{\neg\varphi}), \\
&\quad \pi \models \text{co-Büchi}(L \times Q_{\neg\varphi}) \\
&\iff \mathcal{C} \models \text{co-Büchi}(L \times Q_{\neg\varphi})
\end{aligned}$$

Ceci conclut donc la preuve de ce théorème. □

Il reste à remarquer que l'automate temporisé  $\mathcal{A} \times \mathcal{B}_{\neg\varphi}$  satisfait les restrictions portant sur les horloges bornées et les cycles de l'automate des régions dès lors que  $\mathcal{A}$  les satisfait. Comme nous avons de plus montré dans la section 8.3 comment vérifier des conditions co-Büchi, nous obtenons le corollaire suivant :

**Corollaire 8.24.** *Le problème de la vérification robuste de LTL sur les automates temporisés est décidable et PSPACE-complet.*

Habituellement, la vérification de LTL sur les structures finies est un problème PSPACE-complet, mais dont la complexité n'est que NLOGSPACE dans la taille du système à analyser. Dans le cas des automates temporisés, les problèmes de vérification standard et robuste sont tous deux PSPACE-complets, et le sont à la fois dans la taille de la structure et dans la taille de la formule.

## 8.5 Vers la vérification robuste de MTL

La logique MTL [Koy90, AH93] étend la logique LTL avec des contraintes temporelles quantitatives sur les modalités « jusqu'à ». Nous présentons ici un premier résultat positif vers la vérification

robuste de propriétés exprimées dans la logique MTL. Nous considérons la propriété suivante exprimant un « temps de réponse borné » :

$$\varphi = \mathbf{G} (a \rightarrow \mathbf{F}_{\leq c} b).$$

où  $a$  et  $b$  représentent des actions (éléments de  $\Sigma$ ) et  $c$  appartient à  $\mathbb{T}$  et  $\rightarrow$  représente l'opérateur « implique » habituel. Cette formule exprime que lorsqu'un événement  $a$  intervient, alors il doit être suivi d'un événement  $b$  au plus  $c$  unités de temps plus tard. Cette propriété contraint donc les délais de réaction du système. La satisfaction robuste d'une telle propriété assure donc que le système, même sous de petites perturbations, vérifie cette propriété quantitative donnée par le délai  $c$ . Soulignons que seul le système est « perturbé », les contraintes exprimées par la formule demeurent les mêmes.

**Sémantique.** Afin de définir formellement la satisfaction de  $\varphi$  sur un chemin, nous avons besoin d'informations temporelles sur le chemin. Pour cela, nous définissons pour un chemin  $\pi$  sa longueur temporelle entre deux de ses actions. Considérons donc un chemin infini  $\pi$  :

$$(\ell_0, v_0) \xrightarrow{d_0} (\ell_0, v_0 + d_0) \xrightarrow{\delta_0} (\ell_1, v_1) \cdots (\ell_k, v_k) \xrightarrow{d_k} (\ell_k, v_k + d_k) \xrightarrow{\delta_k} \cdots$$

Étant donnés deux indices  $i_1 < i_2$ , nous définissons la *longueur temporelle de  $\pi$  entre les actions  $\delta_{i_1}$  et  $\delta_{i_2}$* , notée *délai*( $\delta_{i_1}, \delta_{i_2}$ ), comme la valeur  $\sum_{j=i_1+1}^{i_2} d_j$ . Nous disons alors qu'un chemin  $\pi$  satisfait la formule  $\varphi$ , ce qui est noté par  $\pi \models \varphi$ , si et seulement si :

$$\forall i \geq 0, \text{ si } \delta_i = a, \text{ alors } \exists j > i \text{ tel que } \delta_j = b \text{ et } \text{délai}(\delta_i, \delta_j) \leq c.$$

Ceci implique que le chemin  $\pi$  vérifie également la formule de LTL  $\mathbf{G} (a \rightarrow \mathbf{F} b)$ .

### Vérification robuste de propriétés de temps de réponse borné.

**Théorème 8.25.** *La vérification robuste de propriétés de temps de réponse borné sur les automates temporisés est décidable et dans PSPACE.*

**Preuve.** Soit  $\varphi = \mathbf{G} (a \rightarrow \mathbf{F}_{\leq c} b)$ . Nous supposons que  $\mathcal{A}$  est un automate temporisé vérifiant la propriété LTL  $\mathbf{G} (a \rightarrow \mathbf{F} b)$ . Nous démontrons l'équivalence suivante :

$$\mathcal{A} \not\models \varphi \iff \text{il existe un état } \alpha \text{ dans } \text{Acc}^*(\mathcal{A}) \text{ tel qu'il existe un chemin fini dans } \llbracket \mathcal{A} \rrbracket \text{ issu de } \alpha, \text{ commençant par une action } a, \text{ contenant exactement une action } b \text{ en dernière position et tel que le temps écoulé entre ces deux actions est plus grand que } c.$$

- Nous supposons que  $\mathcal{A} \not\models \varphi$ . Pour tout  $\Delta > 0$ , il existe un chemin  $\pi_\Delta$  dans  $\llbracket \mathcal{A} \rrbracket$  qui ne vérifie pas  $\varphi$ . Nous considérons l'ensemble des chemins  $\pi_\Delta$  pour  $\Delta \leq 1$ . Tous ces chemins sont des chemins de  $\mathcal{A}_1$ . Soit  $(\ell_\Delta, v_\Delta)$  la configuration atteinte immédiatement avant que l'action  $a$  témoin de la violation de la satisfaction de  $\varphi$  ait lieu. Soit  $T_\Delta$  la trace entre  $(\ell_\Delta, v_\Delta)$  et la première action  $b$  apparaissant le long de  $\pi_\Delta$  après  $(\ell_\Delta, v_\Delta)$ . Cette première action  $b$  existe nécessairement car nous avons supposé que le système vérifie robustement la propriété LTL  $\mathbf{G} (a \rightarrow \mathbf{F} b)$ . De plus, la longueur temporelle du fragment de  $\pi_\Delta$  mentionné précédemment est strictement plus grande que  $c$  puisque  $\pi_\Delta \not\models \varphi$ . Notons également qu'il n'existe qu'un nombre fini de telles traces  $T_\Delta$  pour l'ensemble des valeurs de  $\Delta$ . En effet, si ces traces étaient en nombre infini, il existerait un cycle dans le graphe des régions entre les actions  $a$  et  $b$ .



Comme  $\mathcal{A} \models \mathbf{G}(a \rightarrow \mathbf{F}b)$ , un tel cycle ne peut pas exister et l'ensemble des traces  $T_\Delta$  est donc fini. De plus, l'ensemble des configurations accessibles  $\text{Acc}(\mathcal{A}_1)$  est compact car il est fermé et borné (et l'espace vectoriel est de dimension finie). Nous avons en effet supposé que les automates temporisés considérés sont bornés et ont des gardes larges. Ainsi, d'après cette propriété de compacité, nous pouvons extraire une sous-séquence infinie  $(\Delta_i)_{i \geq 0}$  ayant pour support la même trace  $T$ , convergeant vers 0 et telle que la séquence  $(\ell_{\Delta_i}, v_{\Delta_i})_{i \geq 0}$  converge vers une configuration  $(\ell, v)$ . De plus, la partie de  $\pi_{\Delta_i}$  allant de la configuration  $(\ell_{\Delta_i}, v_{\Delta_i})$  à la première action  $b$  a une longueur temporelle strictement supérieure à  $c$ . Ceci implique donc les deux faits suivants :

- D'abord, nous avons  $(\ell, v) \in \text{Acc}^*(\mathcal{A})$ . En effet, d'après le théorème 32 de [DDMR04b], pour tout  $\forall \alpha > 0$ , il existe un entier naturel  $i$  tel que pour tout entier naturel  $j \geq i$ , nous avons  $d((\ell_{\Delta_j}, v_{\Delta_j}), \text{Acc}^*(\mathcal{A})) < \alpha$ . Ceci implique  $d((\ell, v), \text{Acc}^*(\mathcal{A})) = 0$  et comme l'ensemble  $\text{Acc}^*(\mathcal{A})$  est fermé (du point de vue topologique), nous obtenons que le point limite  $(\ell, v)$  appartient à  $\text{Acc}^*(\mathcal{A})$ .
- De plus, nous affirmons qu'il existe un chemin réel le long de la trace  $T$  dans  $\llbracket \mathcal{A} \rrbracket$  issu de  $(\ell, v)$ , commençant par une action  $a$ , finissant par une action  $b$ , et de longueur temporelle entre ces deux actions supérieure à  $c$ . En effet, notons que pour toute contrainte (fermée)  $g$ ,

$$(\ell_{\Delta_i}, v_{\Delta_i}) \xrightarrow{i \rightarrow \infty} (\ell, v) \text{ et } (\ell_{\Delta_i}, v_{\Delta_i}) \models_{\Delta_i} g \text{ implique } (\ell, v) \models g.$$

Comme la trace  $T$  est de longueur finie, disons  $k$ , il existe une extraction de la séquence  $(\Delta_i)_{i \geq 0}$ , que nous notons encore  $(\Delta_i)_{i \geq 0}$ , et telle que le long du chemin correspondant à  $T$ , chaque délai des transitions de délai converge<sup>1</sup> : soit  $t_i^j$ , avec  $1 \leq j \leq k$ , le délai associé à la  $j^{\text{e}}$  transition de délai du chemin correspondant à  $\Delta_i$ , nous avons :

$$\forall 1 \leq j \leq k, t_i^j \xrightarrow{i \rightarrow \infty} t^j \quad (8.1)$$

Appliquant la remarque précédente, nous pouvons construire un chemin dans  $\llbracket \mathcal{A} \rrbracket$  comme suit :

$$(\ell, v) \xrightarrow{T_1=a} \xrightarrow{t^1} T_2 \xrightarrow{\dots} \xrightarrow{t^{k-1}} T_k \xrightarrow{t^k} b \xrightarrow{\dots} .$$

Comme affirmé précédemment, ce chemin a une longueur temporelle plus grande (au sens large) que  $c$ . En effet, nous avons pour tous les indices  $i$ ,  $\sum_{j=1}^k t_i^j > c$  ce qui nous donne à la

limite d'après (8.1),  $\sum_{j=1}^k t^j \geq c$ .

- Enfin, ceci conclut donc la preuve de la première implication de l'équivalence.
- Réciproquement, supposons qu'il existe un état  $\alpha$  dans  $\text{Acc}^*(\mathcal{A})$  à partir duquel il existe un chemin fini commençant par une action  $a$ , finissant par un  $b$  et tel que le temps écoulé entre ces deux actions est plus grand que  $c$ . Nous notons  $\rho$  ce chemin témoin. Puisque  $\alpha \in \text{Acc}^*(\mathcal{A})$ , pour tout  $\Delta > 0$ , il existe un chemin  $\pi_\Delta$  de  $(\ell_0, v_0)$  à  $\alpha$ . De plus, pour tout  $\Delta > 0$ , nous pouvons modifier le chemin  $\rho$  de  $\llbracket \mathcal{A} \rrbracket$  en un chemin  $\rho_\Delta$  de  $\llbracket \mathcal{A}_\Delta \rrbracket$  en attendant  $\Delta$  unités de temps supplémentaires immédiatement avant le franchissement de l'action  $b$ . Ceci est possible puisque dans l'automate  $\mathcal{A}_\Delta$ , la garde correspondante a été élargie de  $\Delta$ . Nous obtenons donc par concaténation un chemin  $\pi_\Delta \cdot \rho_\Delta$  de  $\llbracket \mathcal{A}_\Delta \rrbracket$  ne vérifiant pas la propriété  $\varphi$ . Puisque nous avons un tel chemin pour tout  $\Delta > 0$ , nous obtenons  $\mathcal{A} \not\models \varphi$ .

<sup>1</sup>Notons que tous ces délais sont bornés par  $M$  si l'automate temporisé est borné par  $M$ .

Ceci conclut la preuve de la seconde implication.

Le membre droit de l'équivalence précédente est décidable. Une solution consiste à utiliser les « coins » des régions, introduits dans [BFH<sup>+</sup>01], car les chemins de durée maximale passent toujours par de tels points. Un tel algorithme a une complexité PSPACE, ce qui conclut la preuve.  $\square$

**Remarque 8.26.** La preuve précédente est quelque peu *ad-hoc*, puisqu'elle est spécifique à la formule que nous avons considérée. Cependant, elle peut être adaptée par exemple à des propriétés d'invariance comme  $\mathbf{G}(a \rightarrow \mathbf{G}_{\leq c} \neg b)$ . Naturellement, notre souhait est d'étendre les constructions introduites dans ce chapitre de sorte à vérifier robustement un fragment plus grand de la logique MTL.  $\lrcorner$

**Remarque 8.27** (Vérification de MTL). La vérification robuste de l'ensemble de la logique MTL sur les mots temporisés infinis est un problème indécidable. En effet, il est possible de réduire le problème de la validité de MTL (pour les mots infinis) à la vérification robuste de MTL (pour les mots infinis). Considérons une formule  $\varphi \in \text{MTL}$  et l'automate temporisé  $\mathcal{A}$  acceptant l'ensemble des mots infinis. Nous obtenons que  $\varphi$  est valide si et seulement si  $\mathcal{A}$  vérifie robustement  $\varphi$ . Le résultat récent prouvé dans [OW06a] montrant que la satisfaisabilité de MTL sur les mots infinis est un problème indécidable permet de conclure (la validité de  $\varphi$  est équivalente à la satisfaisabilité de  $\neg\varphi$ ).  $\lrcorner$

## 8.6 Conclusion

Dans ces travaux, nous avons étendu les résultats de [DDMR04a] afin de décider une condition suffisante pour l'implémentabilité d'un automate temporisé vis-à-vis d'une propriété de chemin. Dans ce but, nous avons défini une notion de *satisfaction robuste* et proposé des algorithmes de vérification permettant de décider si un automate temporisé vérifie robustement une propriété de chemin. Dans [DDMR04a], le cas des propriétés de sûreté simples a été montré décidable et un algorithme PSPACE proposé pour ce problème. Nous avons développé ici des algorithmes PSPACE pour la vérification robuste de conditions co-Büchi et de propriétés exprimées dans LTL. Nous avons également démontré que ces algorithmes sont optimaux. Enfin, nous avons réalisé un premier pas vers la vérification robuste de propriétés temporisées exprimées dans MTL en présentant un algorithme PSPACE pour des propriétés de temps de réponse borné.

Remarquons que nos résultats peuvent s'étendre à d'autres types de perturbations : dans [Pur98], Puri considère l'introduction de déviations dans les pentes des horloges à la place de l'élargissement des gardes. En réalité, il apparaît que ces deux extensions ont le même effet que les ensembles d'états accessibles [Pur98, DDMR04a, Doy06], et il semble donc naturel que nos preuves peuvent s'étendre au cas des déviations dans les pentes d'horloges.

Par ailleurs, le cas des propriétés de temps de réponse borné est encourageant et laisse penser qu'il est possible de vérifier robustement un ensemble plus grand de propriétés exprimées en MTL. Rappelons toutefois que la vérification robuste de l'ensemble de la logique MTL est indécidable (remarque 8.27). Cependant, des travaux récents [OW06b, BMOW07] ont permis d'exhiber des sous-classes décidables de MTL et dans le cadre des mots finis, les problèmes de satisfaisabilité et de vérification de MTL sont décidables [OW05].

Dans une direction assez proche, il a été démontré dans [AFH96] que la logique MITL, obtenue comme la restriction de MTL interdisant les intervalles ponctuels, est décidable et que sa complexité est plus faible (EXSPACE). Intuitivement, il est souhaitable que la sémantique robuste ne puisse pas détecter de tels intervalles et nous pouvons donc espérer obtenir de la même façon des complexités plus faibles pour les problèmes de vérification robuste que pour les problèmes de vérification classiques.

---

Du point de vue de la sémantique, nous avons considéré dans ces travaux la sémantique ponctuelle de LTL. Il pourrait être intéressant de s'intéresser à la sémantique continue.

Enfin, toutes les propriétés que nous avons considérées sont des propriétés de chemins, *i.e.* des propriétés linéaires. Ceci est dû aux résultats de [DDR04] qui permettent de garantir l'implémentabilité seulement pour ce type de propriétés. Étendre ce cadre de travail à des propriétés arborescentes afin de permettre la vérification robuste de logiques comme CTL ou même TCTL constitue donc un défi important. Ceci pourrait permettre de comparer nos travaux à ceux de [HMP05].



# Conclusion Générale

L'ensemble de ces travaux s'inscrit dans le cadre de la vérification de systèmes temporisés et distribués. Nos contributions portent sur trois aspects : l'étude des formalismes de modélisation, l'algorithmique de la vérification et l'implémentabilité. Nous dressons un bilan transversal de ces contributions, afin d'offrir un point de vue différent. Nous avons vu dans l'introduction qu'il fallait disposer de modèles riches (mais pas trop pour rester analysables) et surtout appropriés aux besoins. L'étude de modèles variés est donc fondamentale, et nous proposons d'apprécier nos travaux du point de vue des modèles et de leurs sémantiques. Nous distinguons quelles sont les caractéristiques des modèles que nous avons étudiés, pourquoi ces caractéristiques sont importantes et expliquons quels sont nos apports, du point de vue de l'expressivité ou de l'algorithmique.

**Transitions silencieuses.** Le rôle des transitions silencieuses dans les automates temporisés est différent de celui qu'elles jouent dans les automates finis. En effet, alors qu'elles n'accroissent pas le pouvoir d'expression de ces derniers, les langages temporisés  $\varepsilon$ -réguliers contiennent strictement les langages temporisés réguliers. Nous avons d'abord démontré que déterminer s'il est possible de s'affranchir des transitions silencieuses dans un automate temporisé est un problème indécidable. Nous avons ensuite étudié la classe des langages temporisés  $\varepsilon$ -réguliers afin de déterminer si elle possède des propriétés que ne possède pas la classe des langages temporisés. Nous nous sommes par exemple intéressés à la notion de complémentation qui fait cruellement défaut aux langages temporisés, ainsi qu'à d'autres problèmes comme la réduction du nombre d'horloges. Nous avons prouvé que tous ces problèmes sont indécidables, démontrant ainsi que la classe des langages temporisés  $\varepsilon$ -réguliers n'est pas plus favorable que la classe des langages temporisés réguliers.

**Gardes diagonales et mises à jour intégrales.** Les gardes diagonales et les mises à jour intégrales, en revanche, n'augmentent pas le pouvoir d'expression des automates temporisés. Cependant, chacune de ces deux extensions est exponentiellement plus concise que le modèle des automates temporisés d'Alur et Dill. Ceci implique qu'analyser des automates temporisés étendus en les transformant explicitement en des automates temporisés d'Alur et Dill conduit à des algorithmes inefficaces. Nous nous sommes donc intéressés au développement d'algorithmes nouveaux adaptés à ces classes d'automates temporisés étendus.

Dans un premier temps, nous avons proposé une traduction efficace de ces automates étendus dans le modèle des réseaux de Petri temporels. Notre traduction est quadratique en général, et linéaire si l'automate ne contient que des gardes diagonales ou que des mises à jour intégrales. Nous évitons ainsi le coût exponentiel mentionné précédemment. Cette transformation permet ensuite d'appliquer les algorithmes existants pour les réseaux de Petri temporels aux objets obtenus. Nous avons ainsi proposé le premier algorithme permettant de traiter de façon simple et uniforme l'ensemble des automates temporisés étendus avec des gardes diagonales et des mises à jour intégrales. Enfin, l'implémentation

de notre traduction dans un prototype a mis en évidence d'une part que la transformation est très rapide mais d'autre part que les modèles obtenus sont difficiles à analyser pour l'outil TINA. Il serait donc nécessaire de dégager des sous-classes de modèles pour lesquelles notre transformation est utilisable en pratique.

Dans un second temps, nous nous sommes intéressés aux automates étendus avec des gardes diagonales. Nous avons rappelé le fonctionnement de l'algorithme classique d'analyse en avant à la volée des automates temporisés et pourquoi il n'est pas correct en présence de gardes diagonales. Cet algorithme est largement utilisé du fait de son efficacité et parce que son parcours en avant (et non en arrière) lui permet de traiter des automates étendus avec des variables entières bornées, ce qui peut être très utile lors de la modélisation. De plus, les automates temporisés avec gardes diagonales pour lesquels la réponse de l'algorithme classique est erronée sont très rares. Notre objectif était donc de développer un algorithme pour analyser les automates temporisés avec gardes diagonales qui ne présente pas de surcoût pour les modèles pour lesquels l'algorithme classique est correct. Cet algorithme doit aussi être intégrable dans un outil de vérification comme UPPAAL. Nous avons pour cela choisi d'utiliser le paradigme du raffinement fondé sur l'analyse des contre-exemples, obtenant ainsi un algorithme satisfaisant ces deux critères. Enfin, l'implémentation de cet algorithme dans l'outil UPPAAL a permis de montrer son efficacité en pratique.

**Arcs de lecture.** Les arcs de lecture constituent une extension habituelle des réseaux de Petri. Alors qu'ils n'augmentent pas l'expressivité du modèle du point de vue de la sémantique des entrelacements, ils permettent de décrire les systèmes de manière plus précise du point de vue d'une sémantique concurrente.

Nous nous sommes d'abord intéressés à l'extension du modèle des réseaux de Petri temporisés à l'aide d'arcs de lecture. Nous avons montré que les automates temporisés et les réseaux de Petri temporisés bornés avec arcs de lecture sont fortement bisimilaires. De plus, la transformation des automates temporisés en réseaux de Petri temporisés saufs avec arcs de lecture est très simple, préserve la taille du modèle et ne modifie pas la structure du système : contrairement aux transformations liant les automates temporisés et les réseaux de Petri temporels, celle-ci n'introduit donc pas de complexité supplémentaire due au codage. Nous avons également démontré que le problème de couverture est décidable pour les réseaux de Petri temporisés avec arcs de lecture. Enfin, nous nous sommes posés la question naturelle du retrait de ces arcs. Alors que pour la sémantique des entrelacements, il est possible de les retirer dans les réseaux de Petri, la situation dans le cadre temporisé est nettement moins simple. Nous avons en effet démontré qu'il est possible de les retirer du point de vue des équivalences de mots finis et de mots infinis non Zeno, mais au prix de transformations très complexes et donc très coûteuses. De plus, du point de vue de l'équivalence générale des mots infinis, *i.e.* en présence de comportements Zeno, il n'est pas possible de s'affranchir des arcs de lecture. Finalement, l'équivalence entre les automates temporisés et les réseaux de Petri temporisés bornés avec arcs de lecture nous a permis, avec ces résultats, d'obtenir une comparaison précise des pouvoirs expressifs des automates temporisés et des réseaux de Petri temporisés.

Par ailleurs, la transformation mentionnée plus haut nous a également suggéré une nouvelle approche pour l'analyse des automates temporisés. Nous avons cherché à développer, à partir de cette transformation, un algorithme permettant de construire un dépliage temporisé d'un réseau d'automates temporisés. Du fait du codage des automates temporisés par des réseaux de Petri temporisés avec arcs de lecture, ce dépliage fait intervenir des arcs de lecture. Ceux-ci nous permettent à la fois d'exprimer les tests d'horloges qui ne sont pas remises à zéro ainsi que les dépendances temporelles liées aux invariants. Nous proposons ainsi une solution aux difficultés majeures liées à la présence de

contraintes temporisées dans un système distribué. Nous avons ensuite décrit comment attacher des zones aux événements de ce dépliage afin de représenter l'ensemble des configurations accessibles dans le réseau, obtenant donc un dépliage temporisé. Nous avons également présenté un algorithme permettant un calcul local du dépliage temporisé. Enfin, nous avons proposé deux approches permettant d'exhiber un préfixe fini et complet de ce dépliage temporisé, rendant ainsi possible l'utilisation de cette construction pour des objectifs de vérification.

**Mises à jour non déterministes.** Les mises à jour non déterministes augmentent le pouvoir d'expression des automates temporisés. Pourtant, le modèle des réseaux de Petri temporisés possède ce type de mises à jour. Nous avons donc étudié leur pouvoir expressif dans ce dernier modèle. Les résultats dépendent en fait de plusieurs critères. Ainsi, dans le modèle standard ne contenant pas d'arcs de lecture, il est possible de les retirer. En revanche, en présence d'arcs de lecture, nous avons à nouveau mis en évidence que les différences d'expressivité sont dues à la présence de comportements Zeno. Dans ce cas, il est en effet nécessaire de raffiner la granularité du modèle afin de pouvoir les simuler par des remises à zéro. Enfin, l'équivalence que nous avons démontrée entre les réseaux de Petri bornés avec arcs de lecture et les automates temporisés nous permet d'obtenir des résultats nouveaux sur le rôle des mises à jour non déterministes dans les automates temporisés.

**Sémantique élargie.** La sémantique traditionnelle des automates temporisés est relativement éloignée des contraintes matérielles réelles. Une sémantique élargie, assouplie, permet de prendre en compte ces contraintes supplémentaires et ainsi de garantir l'implémentabilité du système. Le problème de vérification robuste consiste alors à résoudre le problème de la satisfaction d'une formule par un modèle vis-à-vis de cette sémantique. Nous avons proposé des algorithmes optimaux pour résoudre ce problème pour des conditions co-Büchi et des propriétés exprimées en LTL. Enfin, nous avons également présenté un algorithme permettant de vérifier des propriétés de temps de réponse borné.

## Prolongements

Nous avons déjà mentionné, à l'issue de chaque chapitre, les différents prolongements possibles à chacun de nos travaux. Nous donnons ici plusieurs grandes perspectives plus générales qui nous intéressent particulièrement.

**À la recherche du bon modèle.** Deux objectifs principaux se dégagent, qui tous les deux sont liés à la notion d'urgence. D'abord, il serait intéressant de parvenir à proposer une extension temporisée des réseaux de Petri vérifiant les deux propriétés suivantes :

1. intégrer un mécanisme permettant d'exprimer une notion d'urgence,
2. avoir un problème de couverture décidable (pour les réseaux non bornés).

Rappelons que les réseaux de Petri temporels, dont la sémantique est forte, ont un problème de couverture indécidable. Au contraire, le problème de couverture est décidable pour les réseaux de Petri temporisés, mais leur sémantique est faible. Mentionnons également le modèle des réseaux de systèmes temporisés identiques, introduit dans [ADM04]. Ce modèle permet de vérifier des réseaux non bornés simples, mais ne dispose pas d'urgence.

Dans une autre direction, il nous semble intéressant de chercher à étendre les liens existant entre les automates temporisés et les réseaux de Petri temporisés avec arcs de lecture afin d'obtenir des

relations de bisimulation. Pour cela, il est nécessaire d'introduire de l'urgence dans ce dernier modèle. Une solution consiste à imiter le mécanisme des invariants des automates temporisés, mais il est facile de constater que cela introduit de nombreux blocages temporels. Plus généralement, l'objectif est ici de définir un modèle intermédiaire entre les réseaux de Petri temporisés avec arcs de lecture et les réseaux d'automates temporisés avec invariants et horloges partagées possédant des caractérisations simples dans chacun de ces deux modèles. Celles-ci permettraient de transférer les méthodes d'un modèle à l'autre de façon efficace. Ainsi, l'algorithme que nous avons proposé pour les dépliages s'adapterait facilement au cadre des réseaux de Petri temporisés avec arcs de lecture.

**Poursuivre nos travaux sur les dépliages.** Nos travaux peuvent être poursuivis suivant deux directions qui dépendent de l'objectif dans lequel le dépliage est construit. Ainsi, pour l'observation ou le diagnostic, il est important que ce dépliage soit précis. Or nous avons vu que notre dépliage, par rapport à celui construit dans [CCJ06], peut parfois être moins concis mais que notre algorithme est nettement plus efficace. Il serait donc intéressant de trouver un juste compromis entre ces deux critères. Une solution consiste à chercher des optimisations à notre construction permettant de réduire certaines dépendances. Nous avons par exemple déjà obtenu un critère portant sur la nature des invariants locaux à la transition permettant d'éviter la construction de certaines copies d'événements. Ceci permet, à moindre coût, de réduire la taille de notre dépliage en évitant certaines redondances.

Si au contraire l'objectif de la construction du dépliage est, par exemple, la vérification d'une propriété d'accessibilité ou de sûreté, alors il peut être judicieux d'introduire des approximations dans notre construction. Nous avons utilisé des abstractions pour les gardes diagonales dans le chapitre 6 afin d'éviter un calcul précis qui aurait été trop coûteux. Ceci permet alors l'utilisation de l'approche « CEGAR » pour la construction du dépliage. Il est même envisageable d'utiliser l'optimisation introduite dans [HJMS02] consistant à réutiliser autant que possible les objets déjà construits lors des approximations précédentes. Au lieu de construire un dépliage représentant exactement l'ensemble des configurations accessibles, il peut être suffisant d'être précis seulement pour certains automates du réseau par exemple. Notons enfin que cette méthode pourrait également être appliquée pour la construction de dépliages de systèmes non temporisés.

**Implémentabilité.** L'approche fondée sur la sémantique AASAP nécessite de développer de nouveaux algorithmes de vérification puisque la sémantique est modifiée. D'après les résultats obtenus, les nouveaux problèmes de vérification ont une complexité semblable à celle des problèmes associés à la sémantique classique. Au contraire, l'approche fondée sur la modélisation [AT05] peut réutiliser les travaux existants pour la sémantique classique mais la modélisation explicite de la plate-forme rend les problèmes de vérification plus complexes. Il serait intéressant de dégager des conditions syntaxiques ou sémantiques, facilement décidables, qui assureraient le transfert de propriétés de la sémantique classique à la sémantique AASAP. Ce critère permettrait de combiner les avantages des deux approches, des problèmes de complexité raisonnable et le bénéfice des travaux existants.



# Bibliographie

- [ACM02] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of the ACM*, 49(2) :172–206, 2002.
- [AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.
- [AD94] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.
- [ADI03] Rajeev Alur, Thao Dang, and Franjo Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 208–223. Springer, 2003.
- [ADM04] Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'04)*, pages 345–354. IEEE Computer Society, 2004.
- [AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.
- [AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1) :116–146, 1996.
- [AFM<sup>+</sup>03] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul pettersson, and Wang Yi. TIMES : A tool for schedulability analysis and code generation of real-time systems. In *Proc. 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2003.
- [AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics : Complexity and expressiveness. *Information and Computation*, 104(1) :35–77, 1993.
- [AL91] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2) :253–284, 1991.
- [AL00] Tuomas Aura and Johan Lilius. A causal semantics for time Petri nets. *Theoretical Computer Science*, 243(1–2) :409–447, 2000.
- [ALM05] Rajeev Alur, Salvatore La Torre, and P. Madhusudan. Perturbed timed automata. In *Proc. 8th International Workshop on Hybrid Systems : Computation and Control (HSCC'05)*, volume 3414 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2005.

- [AM04] Rajeev Alur and P. Madhusudan. Decision problems for timed automata : A survey. In *Proc. 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems : Real Time (SFM-04 :RT)*, volume 3185 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.
- [AMM07] Parosh A. Abdulla, Pritha Mahata, and Richard Mayr. Dense-timed Petri nets : Checking zenoness, token liveness and boundedness. *Logical Methods in Computer Science*, 3(1) :1–61, 2007.
- [AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier Science, 1998.
- [AN01] Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and bqos. In *Proc. 22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2001.
- [Ari96] Ariane 5 – flight 501 failure, 1996. Report by the Inquiry Board, chairman : J-L Lions.
- [Arn94] André Arnold. *Finite Transition System*. Prentice Hall, 1994.
- [Asa04] Eugene Asarin. Challenges in timed languages : From applied theory to basic theory. *The Bulletin of the European Association for Theoretical Computer Science*, 83 :106–120, 2004.
- [AT05] Karine Altisen and Stavros Tripakis. Implementation of timed automata : An issue of semantics or modeling ? In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2005.
- [BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.
- [BC07] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 2007. To appear.
- [BCG88] Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59 :115–131, 1988.
- [BCH<sup>+</sup>05a] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of different semantics for time Petri nets. In *Proc. 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA'05)*, volume 3707 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2005.
- [BCH<sup>+</sup>05b] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of expressiveness for timed automata and time Petri nets. Research Report RI-2005-3, IRCCyN/CNRS, France, 2005.
- [BCH<sup>+</sup>05c] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of the expressiveness of timed automata and time Petri nets. In *Proc. 3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 2005.

- [BCH<sup>+</sup>05d] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. When are timed automata weakly timed bisimilar to time Petri nets? In *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2005.
- [BD91] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions in Software Engineering*, 17(3) :259–273, 1991.
- [BD99] Marc Boyer and Michel Diaz. Non equivalence between time Petri nets and time stream Petri nets. In *Proc. 8th International Workshop on Petri Nets and Performance Models (PNPM'99)*, pages 198–207, 1999.
- [BD01] Marc Boyer and Michel Diaz. Multiple enabledness of transitions in Petri nets with time. In *Proc. 9th International Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 219–228, 2001.
- [BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2–3) :291–345, 2004.
- [BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3) :145–182, 1998.
- [Ben02] Johan Bengtsson. *Clocks, DBMs ans States in Timed Systems*. PhD thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2002.
- [BFH<sup>+</sup>01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th International Workshop on Hybrid Systems : Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.
- [BHR06a] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Extended timed automata and time Petri nets. In *Proc. 6th International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 91–100. IEEE Computer Society Press, 2006.
- [BHR06b] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed Petri nets and timed automata : On the discriminating power of Zeno sequences. In *Proc. 33rd International Colloquium on Automata, Languages and Programming (ICALP'06)*, volume 4052 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006.
- [BHR06c] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed Petri nets and timed automata : On the discriminating power of zeno sequences. Research Report LSV-06-06, Laboratoire Spécification et Vérification, ENS Cachan, France, 2006.
- [BHR06d] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed unfoldings for networks of timed automata. In *Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 292–306. Springer, 2006.
- [BHR07] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Undecidability results for timed automata with silent transitions. Research Report LSV-07-12, Laboratoire Spécification et Vérification, ENS Cachan, France, 2007.

- [BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 1998.
- [BLR05] Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata : Forward analysis of timed systems. In *Proc. 3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2005.
- [BLT90] Tommaso Bolognesi, Ferdinando Lucidi, and Sebastiano Trigila. From timed Petri nets to timed LOTOS. In *Proc. 10th IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification (PSTV'90)*, pages 395–408. North-Holland, 1990.
- [BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In *Proc. 9th IFIP Congress*, pages 41–46. Elsevier Science Publishers, 1983.
- [BMOW07] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. The cost of punctuality. In *Proc. 22nd Annual Symposium on Logic in Computer Science (LICS'07)*, Wroclaw, Poland, 2007. IEEE Computer Society Press. To appear.
- [BMR06] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust model-checking of linear-time properties in timed automata. In *Proc. 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, volume 3887 of *Lecture Notes in Computer Science*, pages 238–249, Valdivia, Chile, 2006. Springer.
- [Bou02] Patricia Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2) :75–85, 2002.
- [Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3) :281–320, 2004.
- [BP02] Patricia Bouyer and Antoine Petit. A Kleene/Büchi-like theorem for clock languages. *Journal of Automata, Languages and Combinatorics*, 7(2) :167–186, 2002.
- [BPT02] Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 2002.
- [BPV06] Bernard Berthomieu, Florent Peres, and François Vernadat. Bridging the gap between timed automata and bounded time petri nets. In *Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2006.
- [BR07] Marc Boyer and Olivier H. Roux. Comparison of the expressiveness of arc, place and transition time Petri nets. In *Proc. 28th International Conference on Application and Theory of Petri Nets (ICATPN'07)*, volume 4546 of *Lecture Notes in Computer Science*, pages 63–82. Springer, 2007.
- [Bre06] The machinery of democracy : Protecting elections in an electronic world, 2006. Report by Brennan Center Task Force on Voting System Security, Lawrence Norden, Chair.
- [BRV04] Bernard Berthomieu, Pierre-Olivier Ribet, and François Vernadat. Construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14) :2741–2756, 2004.

- [BSBM06] Ramzi Ben Salah, Marius Bozga, and Oded Maler. On interleaving in timed automata. In *Proc. 17th International Conference on Concurrency Theory (CONCUR'06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2006.
- [BV03] Bernard Berthomieu and François Vernadat. State class constructions for branching analysis of time Petri nets. In *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 442–457. Springer, 2003.
- [BY03] Johan Bengtsson and Wang Yi. On clock difference constraints and termination in reachability analysis of timed automata. In *Proc. 5th International Conference on Formal Engineering Methods (ICFEM 2003)*, volume 2885 of *Lecture Notes in Computer Science*, pages 491–503. Springer, 2003.
- [CCJ06] Franck Cassez, Thomas Chatain, and Claude Jard. Symbolic unfoldings for networks of timed automata. In *Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 2006.
- [CDP06] Fabrice Chevalier, Deepak D'Souza, and Pavithra Prabakhar. On continuous timed automata with input-determined guards. In *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 2006.
- [CGJ<sup>+</sup>00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proc. 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.
- [CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model-Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
- [CHR02] Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In *Proc. 5th International Workshop on Hybrid Systems : Computation and Control (HSCC'02)*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2002.
- [CJ05] Thomas Chatain and Claude Jard. Time supervision of concurrent systems using symbolic unfoldings of time Petri nets. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2005.
- [CJ06] Thomas Chatain and Claude Jard. Complete finite prefixes of symbolic unfoldings of time Petri nets. In *Proc. 27th International Conference on Application and Theory of Petri Nets (ICATPN'06)*, volume 4024 of *Lecture Notes in Computer Science*, pages 125–145. Springer, 2006.
- [CLM76] E. Cardoza, Richard J. Lipton, and Albert R. Meyer. Exponential space complete problems for Petri nets and commutative semigroups : Preliminary report. In *Proc. 8th Annual ACM Symposium on Theory of Computing*, pages 50–54. ACM, 1976.
- [CR03] Franck Cassez and Olivier H. Roux. Traduction structurelle des réseaux de Petri temporels vers les automates temporisés. In *Actes 4ième Colloque sur la Modélisation des Systèmes Réactifs (MSR'03)*, pages 311–326. Hermès, 2003.

- [CR06] Franck Cassez and Olivier H. Roux. Structural translation from Time Petri Nets to Timed Automata – Model-Checking Time Petri Nets via Timed Automata. *The Journal of Systems and Software*, 79(10) :1456–1468, 2006.
- [DDMR04a] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robustness and implementability of timed automata. In *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2004.
- [DDMR04b] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robustness and implementability of timed automata. Technical Report 2004.30, Centre Fédéré en Vérification, Belgium, 2004.
- [DDR04] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics : From timed models to timed implementations. In *Proc. 7th International Workshop on Hybrid Systems : Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2004.
- [DDR05a] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost asap semantics : from timed models to timed implementations. *Formal Aspects of Computing*, 17(3) :319–341, 2005.
- [DDR05b] Martin De Wulf, Martin Doyen, and Jean-François Raskin. Systematic implementation of real-time models. In *Proc. Formal Methods (FM'05)*, volume 3582 of *Lecture Notes in Computer Science*, pages 139–156. Springer, 2005.
- [DDSS07] Davide D'Aprile, Susanna Donatelli, Arnaud Sangnier, and Jeremy Sproston. From time Petri nets to timed automata : an untimed approach. In *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2007.
- [dFERA00] David de Frutos-Escrig, Valentín Valero Ruiz, and Olga Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *Proc. 21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*, pages 187–206, 2000.
- [Dil90] David Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.
- [Dim99] Cătălin Dima. Kleene theorems for event-clock automata. In *Proc. 12th International Symposium on Fundamentals of Computation Theory (FCT'99)*, volume 1684 of *Lecture Notes in Computer Science*, pages 215–225. Springer, 1999.
- [Dim01] Cătălin Dima. Real-time automata. *Journal of Automata, Languages and Combinatorics*, 6(1) :3–24, 2001.
- [Dim05] Cătălin Dima. Timed shuffle expressions. In *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2005.
- [DK06] Conrado Daws and Piotr Kordy. Symbolic robustness analysis of timed automata. In *Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2006.

- [Doy06] Laurent Doyen. *Algorithmic Analysis of Complex Semantics for Timed and Hybrid Automata*. PhD thesis, Université Libre de Bruxelles, Belgium, 2006.
- [D'S00] Deepak D'Souza. A logical characterization of event recording automata. In *Proc. 9th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00)*, volume 1926 of *Lecture Notes in Computer Science*, pages 240–251. Springer, 2000.
- [DT98] Conrado Daws and Stavros Tripakis. Model-checking of real-time reachability properties using abstractions. In *Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.
- [DW07] Martin De Wulf. *From Timed Models to Timed Implementations*. PhD thesis, Université Libre de Bruxelles, Belgium, 2007.
- [EN94] Javier Esparza and Mogens Nielsen. Decibility issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics*, 30(3) :143–160, 1994.
- [ER99] Javier Esparza and Stefan Römer. An unfolding algorithm for synchronous products of transition systems. In *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 1999.
- [ERV02] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3) :285–310, 2002.
- [Fin06] Olivier Finkel. Undecidable problems about timed automata. In *Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2006.
- [FPY02] Elena Fersman, Paul Petterson, and Wang Yi. Timed automata with asynchronous processes : Schedulability and decidability. In *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.
- [FS02] Hans Fleischhack and Christian Stehno. Computing a finite prefix of a time Petri net. In *Proc. 23rd International Conference on Application and Theory of Petri Nets (ICATPN'02)*, volume 2369 of *Lecture Notes in Computer Science*, pages 163–181. Springer, 2002.
- [GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 1997.
- [GLMR05] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Romeo : A tool for analyzing time Petri nets. In *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423. Springer, 2005.
- [GRR03] Guillaume Gardey, Olivier H. Roux, and Olivier F. Roux. A zone-based method for computing the state space of a time Petri net. In *Proc. 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 246–259. Springer, 2003.

- [HHK03] Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. GIOTTO : A time-triggered language for embedded programming. *Proc. of the IEEE*, 91(1) :84–99, 2003.
- [Hig52] Graham Higman. Ordering by divisibility in abstract algebras. In *Proc. London Mathematical Society*, volume 2, pages 326–336, 1952.
- [HJMS02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In *Proc. 29th ACM Symposium on Principles of Programming Languages (POPL'02)*, pages 58–70. ACM, 2002.
- [HKWT95] Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995.
- [HMP05] Thomas A. Henzinger, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying similarities between timed systems. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 226–241. Springer, 2005.
- [HR00] Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems. In *Proc. 3rd International Workshop on Hybrid Systems : Computation and Control (HSCC'00)*, volume 1790 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2000.
- [HSLKT02] Stefan Haar, Françoise Simonot-Lion, Laurent Kaiser, and Joël Toussaint. Equivalence of timed state machines and safe time Petri nets. In *Proc. 6th International Workshop on Discrete Event Systems (WoDES'02)*, pages 119–126, 2002.
- [HSSL97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal modeling and analysis of an audio/video protocol : An industrial case study using UPPAAL. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 2–13. IEEE Computer Society Press, 1997.
- [HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
- [JLL77] Neil D. Jones, Lawrence H. Landweber, and Y. Edmund Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4(3) :277–299, 1977.
- [KLL<sup>+</sup>97] Kåre J. Kristoffersen, François Laroussinie, Kim G. Larsen, Paul Pettersson, and Wang Yi. A compositional proof of a real-time mutual exclusion protocol. In *Proc. 7th International Joint Conference on Theory and Practice of Software Development (TAPSOFT'97)*, volume 1214 of *Lecture Notes in Computer Science*, pages 565–579. Springer, 1997.
- [KM69] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2) :147–195, 1969.
- [Kos82] S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14th ACM Symposium Theory of Computing (STOC'82)*, pages 267–281. ACM, 1982.
- [Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4) :255–299, 1990.
- [Lil98] Johan Lilius. Efficient state space search for time Petri nets. In *Electronic Notes in Theoretical Computer Science*, volume 18, 1998.



- [LMS04] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2004.
- [LNZ04] Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. In *Proc. 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 296–311. Springer, 2004.
- [LR03] Didier Lime and Olivier H. Roux. State class timed automaton of a time Petri net. In *Proc. 10th International Workshop on Petri Nets and Performance Models (PNPM'03)*, pages 124–133. IEEE Computer Society Press, 2003.
- [LR06] Didier Lime and Olivier H. Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Events Dynamic Systems - Theory and Applications (DEDS)*, 16(2) :179–205, 2006.
- [LT93] Nancy G. Leveson and Clark S. Turner. Investigation of the therac-25 accidents. *IEEE Computer*, 26(7) :18–41, 1993.
- [LW05] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. In *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.
- [LW07] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic*, 2007. To appear.
- [Mah05] Pritha Mahata. *Model Checking Parameterized Timed Systems*. PhD thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2005.
- [May84] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13(3) :441–460, 1984.
- [McM95] Kenneth McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1) :45–65, 1995.
- [Mer74] Philip M. Merlin. *A Study of the Recoverability of Computing Systems*. PhD thesis, University of California, Irvine, CA, USA, 1974.
- [Min67] Marvin Minsky. *Computation : Finite and Infinite Machines*. Prentice Hall International, 1967.
- [Min99] Marius Minea. Partial order reduction for model checking of timed automata. In *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 1999.
- [MP04] Oded Maler and Amir Pnueli. On recognizable timed languages. In *Proc. 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'04)*, volume 2987 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 2004.
- [MR95] Ugo Montanari and Francesca Rossi. Contextual nets. *Acta Informatica*, 32(6) :545–596, 1995.
- [NPW81] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13 :85–108, 1981.

- [NQ06a] Peter Niebert and Hongyang Qu. Adding invariants to event zone automata. In *Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2006.
- [NQ06b] Peter Niebert and Hongyang Qu. The implementation of Mazurkiewicz traces in POEM. In *Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2006.
- [OW03] Joël Ouaknine and James B. Worrell. Revisiting digitization, robustness and decidability for timed automata. In *Proc. 18th Annual Symposium on Logic in Computer Science (LICS'03)*. IEEE Computer Society Press, 2003.
- [OW04] Joël Ouaknine and James B. Worrell. On the language inclusion problem for timed automata : Closing a decidability gap. In *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'04)*, pages 54–63. IEEE Computer Society Press, 2004.
- [OW05] Joël Ouaknine and James B. Worrell. On the decidability of metric temporal logic. In *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 188–197. IEEE Computer Society Press, 2005.
- [OW06a] Joël Ouaknine and James B. Worrell. On metric temporal logic and faulty Turing machines. In *Proc. 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.
- [OW06b] Joël Ouaknine and James B. Worrell. Safety metric temporal logic is fully decidable. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2006.
- [Pel93] Doron Peled. All from one, one for all : on model checking using representatives. In *Proc. 5th International Conference on Computer Aided Verification (CAV'93)*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993.
- [Pet62] Carl A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.
- [Pnu77] Amir Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.
- [Pur98] Anuj Puri. Dynamical properties of timed automata. In *Proc. 5th International Symposium on Formal techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, volume 1486 of *Lecture Notes in Computer Science*, pages 210–227. Springer, 1998.
- [Rac78] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6 :223–231, 1978.
- [Ras99] Jean-François Raskin. *Logics, Automata and Classical Theories for Deciding Real-Time*. PhD thesis, University of Namur, Namur, Belgium, 1999.
- [Rey07] Pierre-Alain Reynier. Diagonal constraints handled efficiently in UPPAAL. Research Report LSV-07-02, Laboratoire Spécification et Vérification, ENS Cachan, France, 2007.

- [Srb05] Jiri Srba. Timed-arc Petri nets vs. networks of timed automata. In *Proc. 26th International Conference Application and Theory of Petri Nets (ICATPN'05)*, volume 3536 of *Lecture Notes in Computer Science*, pages 385–402. Springer, 2005.
- [Tri98] Stavros Tripakis. *L'analyse formelle des systèmes temporisés en pratique*. PhD thesis, Université Joseph Fourier, Grenoble, France, 1998.
- [Tri03] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. In *Proc. 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 182–188. Springer, 2003.
- [TSLT97] Joël Toussaint, Françoise Simonot-Lion, and Jean-Pierre Thomesse. Time constraints verification methods based on time Petri nets. In *Proc. 6th IEEE Workshop on Future Trends of Distributed Computer Systems (FTDCS '97)*, pages 262–269. IEEE Computer Society, 1997.
- [TY98] Stavros Tripakis and Sergio Yovine. Verification of the fast reservation protocol with delayed transmission using the tool KRONOS. In *Proc. 4th IEEE Real-Time Technology and Applications Symposium (RTAS'98)*, pages 165–170. IEEE Computer Society Press, 1998.
- [Val89] Antti Valmari. Stubborn sets for reduced state space generation. In *Proc. 10th International Conference on Applications and Theory of Petri Nets (ICATPN'89)*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer, 1989.
- [Var96] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proc. Logics for Concurrency : Structure versus Automata, 8th Banff Higher Order Workshop*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.
- [VFEC99] Valentín Valero, David Frutos-Escrig, and Fernando Cuartero. On non-decidability of reachability for timed-arc Petri nets. In *Proc. 8th International Workshop on Petri Nets and Performance Models (PNPM'99)*, pages 188–196. IEEE Computer Society Press, 1999.
- [Vot07] E-voting failures in the 2006 mid-term election, 2007. Report prepared by VotersUnite.org.
- [VSY98] Walter Vogler, Alexei L. Semenov, and Alexandre Yakovlev. Unfolding and finite prefix for nets with read arcs. In *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 501–516. Springer, 1998.
- [Wil94] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer, 1994.
- [Win02] Józef Winkowski. Reachability in contextual nets. *Fundamenta Informaticae*, 51(1-2) :235–250, 2002.
- [WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems*. PhD thesis, Stanford University, USA, 1994.
- [WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proc. 24th Annual Symposium on Foundations of Computer Science (FOCS'83)*, pages 185–194. IEEE Computer Society Press, 1983.

- [YS97] Tomohiro Yoneda and Bernd-Holger Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 11(2) :187–215, 1997.
- [Zbr05] Andrzej Zbrzezny. SAT-based reachability checking for timed automata with diagonal constraints. *Fundamenta Informaticae*, 67(1–3) :303–322, 2005.