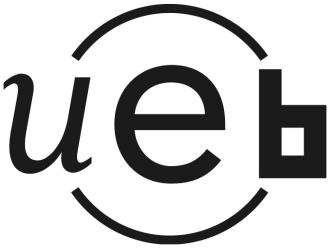
ANNÉE 2013





THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique
Ecole doctorale MATISSE

présentée par

Amélie Stainer

Préparée à l'unité de recherche n°6074 IRISA
Institut de recherche en informatique et systèmes aléatoires
ISTIC

Contribution à la vérification des automates temporisés : déterminisation, vérification quantitative et accessibilité dans les réseaux d'automates

Thèse soutenue à Rennes
le 25 novembre 2013

devant le jury composé de :

Sophie Pinchinat
Professeur à l'Université de Rennes 1 / *présidente du jury*

Paul Gastin
Professeur à l'ENS Cachan / *rapporteur*

Joël Ouaknine
Professeur à l'Université d'Oxford / *rapporteur*

Patricia Bouyer-Decître
Directrice de recherche CNRS, LSV, ENS Cachan / *examinatrice*

Didier Lime
Maître de conférence à Centrale Nantes / *examinateur*

Thierry Jéron
Directreur de recherche à INRIA Rennes-Bretagne Atlantique / *directeur de thèse*

Nathalie Bertrand
Chargée de recherche à INRIA Rennes-Bretagne Atlantique */ co-encadrante de thèse*

Contribution to the Verification of Timed Automata: Determinization, Quantitative Verification and Reachability in Networks of Automata

Amélie Stainer
University of Rennes 1

Supervised by: Nathalie Bertrand and Thierry Jéron
INRIA Rennes - Bretagne atlantique

Rennes - Septembre 2013

Contents

I Contribution à la vérification des automates temporisés **9**

0.1 Introduction à la vérification des automates temporisés 11
0.2 Déterminisation des automates temporisés et application au test 14
0.3 Fréquences dans les automates temporisés 15
0.4 Accessibilité dans les automates temporisés communicants 17
0.5 Conclusion 18

II Introduction **19**

1 Introduction **21**

2 Technical preliminaries **29**

III Determinization of Timed Automata **35**

3 A Game Approach to Determinize Timed Automata **41**

3.1 The game approach 42
3.1.1 Game definition 42
3.1.2 Example 45
3.1.3 Properties of the strategies 47
3.2 Comparison both existing methods 54
3.2.1 Comparison with [KT09] 54
3.2.2 Comparison with [BBBB09] 56
3.3 Extension to ε-transitions and invariants 59
3.3.1 ε-transitions 59
3.3.2 Invariants 61
3.3.3 Properties of the strategies in the extended game 63
3.3.4 Comparison with [KT09] 66
3.4 Beyond over-approximation 67
3.4.1 Under-approximation 67
3.4.2 Combining over- and under-approximation 67
3.5 Implementation of a prototype tool 73
3.5.1 Using zones instead of regions 73
3.5.2 Implementation of the prototype 73
3.5.3 Execution of the program 75

4 Application of the Game Approach to Off-line Test Selection **79**
4.1 A model of open timed automata with inputs / outputs . . . 81
4.1.1 Timed automata with inputs/outputs . . . 81
4.1.2 The semantics of OTAIOs . . . 82
4.1.3 Properties and operations . . . 84
4.2 Conformance testing theory . . . 87
4.2.1 The **tioco** conformance theory . . . 87
4.2.2 Refinement preserving **tioco** . . . 90
4.3 Off-line test case generation . . . 92
4.3.1 Test purposes . . . 92
4.3.2 Principle of test generation . . . 93
4.3.3 Test suite properties . . . 99
4.4 Discussion and related work . . . 102

IV Frequencies in Timed Automata **107**

5 Preliminaries **115**
5.1 Frequencies in timed automata . . . 116
5.2 Frequency-based semantics . . . 118
5.2.1 Frequencies and timed automata . . . 118
5.2.2 A brief comparison with the usual semantics . . . 119
5.2.3 A particular case of double-priced timed automata . . . 120
5.3 The corner-point abstraction . . . 121
5.3.1 Definition and examples . . . 121
5.3.2 Ratios in the corner-point abstraction . . . 124
5.3.3 Set of ratios in the corner-point abstraction . . . 125
5.4 Forgetfulness . . . 128
5.4.1 Forgetfulness and aperiodicity . . . 128
5.4.2 Comparison with the forgetfulness of [BA11] . . . 131

6 Frequencies in One-Clock Timed Automata **133**
6.1 From $\mathcal{A}$ to $\mathcal{A}_{cp}$ . . . 133
6.1.1 Contraction and dilatation . . . 134
6.1.2 Proposition 6.1 does not extend to timed automata with two clocks . . . 138
6.2 From $\mathcal{A}_{cp}$ to $\mathcal{A}$ . . . 139
6.2.1 Reward-diverging case . . . 139
6.2.2 Proposition 6.2 extends neither to timed automata with two clocks, nor to Zeno runs. . . . 140
6.2.3 Proposition 6.2 does not extend to reward-converging runs . . . 141
6.2.4 Reward-converging case . . . 141
6.3 Set of frequencies in $\mathcal{A}$ . . . 142
6.3.1 Set of frequencies of non-Zeno runs in $\mathcal{A}$ . . . 142
6.3.2 Realizability of bounds by Zeno runs in $\mathcal{A}$ . . . 143
6.3.3 Proof of Theorem 6.1 . . . 146

7 Frequencies in Forgetful Timed Automata **149**
7.1 Frequencies in one-clock forgetful timed automata . . . 150
7.2 Extension to several clock forgetful timed automata . . . 153
7.2.1 Inclusion of the set of frequencies in the set of ratios . . . 154
7.2.2 Techniques to compute the frequencies . . . 155
7.2.3 Inclusion of the set of ratios in the set of frequencies . . . 157
7.2.4 Discussion about assumptions . . . 159

8 Emptiness and Universality Problems in Timed Automata with Frequency **161**
8.1 Consequences of Chapters 6 and 7 . . . 161
8.2 Lower bound for the universality problem . . . 162
8.3 Decidability of the universality problem for Zeno words with positive frequency in one-clock timed automata . . . 163

V Reachability of Communicating Timed Automata **167**

9 Communicating Timed Processes: a Uniform Semantics **175**
9.1 Definition of communicating timed processes . . . 175
9.2 Communicating timed or tick automata . . . 177
9.2.1 Communicating timed automata . . . 177
9.2.2 Communicating tick automata . . . 178
9.3 Discussion about the models . . . 179
9.3.1 Modeling urgency with emptiness test . . . 179
9.3.2 On the power of time . . . 180
9.3.3 Undecidability of multi-tick automata . . . 181

10 Reachability Problem in Communicating Tick Automata **183**
10.1 Communicating counter automata . . . 183
10.2 From tick automata to counter automata. . . . 184
10.3 From counter automata to tick automata . . . 187
10.4 Characterization of the decidable topologies . . . 189

11 Reachability Problem in Communicating Timed Automata **191**
11.1 From continuous time to discrete time . . . 191
11.2 Correctness of the reduction . . . 193
11.2.1 Proof of the correctness using a rescheduling lemma . . . 193
11.2.2 Proof of the rescheduling lemma . . . 195
11.2.3 Consequences . . . 198
11.3 Reciprocal reduction and its consequences . . . 198
11.4 Abstraction of communicating timed automata with emptiness tests is difficult . . . 199
11.4.1 Our construction is not sound for emptiness test . . . 199
11.4.2 Why soundness is hard to achieve . . . 200

VI Conclusion and Future Works **203**

Bibliography **213**

Remerciements

Je tiens tout d'abord à remercier mes rapporteurs, Paul Gastin et Joël Ouaknine et plus généralement tous les membres de mon jury, Patricia Bouyer-Decître, Didier Lime et Sophie Pinchinat. Je remercie tous les collègues avec qui j'ai travaillé dans le monde de la recherche comme dans le cadre des enseignements. Je remercie Thierry et Nathalie pour m'avoir supportée durant ces 3 ans et encore aujourd'hui. Ils étaient présents dans les moments festifs comme dans les moments difficiles. Je tiens à remercier tous les membres de l'équipe VerTeCs et plus largement les collègues avec qui nous prenions les pauses. Ces moments étaient une source de joie au quotidien et d'un soutien crucial dans les périodes compliquées. Je remercie ma famille pour tout le bonheur qu'ils me procurent chaque jours. Finalement, je voudrais dire à Philippe tout l'amour que je lui porte, à quel point je suis heureuse de partager sa vie et d'être la maman de notre petit Tao.

Part I

Contributions à la vérification des automates temporisés : déterminisation, vérification quantitative et accessibilité dans les réseaux d'automates

0.1 Introduction à la vérification des automates temporisés

Nous rencontrons de plus en plus de systèmes informatiques dans la vie de tous les jours comme dans les environnements plus spécialisés. Ces systèmes doivent respecter des spécifications, c'est-à-dire avoir un comportement conforme aux attentes. En particulier, beaucoup d'entre eux doivent aussi satisfaire des contraintes sur leur temps de réponse. Ces systèmes sont dits "temps réel". La correction de tels systèmes dépend donc de la validité des sorties du système, mais aussi des délais dans lesquels elles sont émises. Les unités de temps considérées et la précision des horloges dépendent des contextes et peuvent être très différentes. Un premier exemple de contexte d'application des systèmes temps réel est l'industrie aéronautique. Dans l'avionique comme dans l'aérospatial, des ordinateurs sont utilisés comme assistants et contrôleurs durant les vols d'avions ou de navettes. Par ailleurs, l'exploration des planètes est faite par des robots qui collectent et analysent des données. Dans ces situations, les systèmes doivent être réactifs, ne serait-ce que dans la gestion de leurs déplacements. Dans de tels contextes, des erreurs peuvent causer des pertes humaines ou avoir des conséquences économiques considérables. Les systèmes temps réel sont également très utilisés pour la production supervisée dans l'industrie. La production d'éléments chimiques tels que l'éthylène ou le prophylène est ainsi gérée et contrôlée par des ordinateurs. Ces systèmes requièrent une grande précision dans les délais d'actions. La moindre erreur pourrait être à l'origine d'une catastrophe écologique.

Vérification de systèmes temps réel La liste des applications critiques des systèmes temps réel est longue et fournit autant de raisons de développer des méthodes de vérification. Les propriétés attendues pour un système temps réel peuvent être représentées de plusieurs façons. Suivant le niveau de criticité, la connaissance du système et les propriétés à vérifier, beaucoup d'approches sont possibles. Si le système est inconnu, des cas de test peuvent être générés pour interagir avec lui et détecter d'éventuelles non-conformités vis-à-vis de la spécification. En revanche, si le système est connu, la vérification peut être faite à différents niveaux d'abstraction. Des programmes peuvent être vérifiés et il existe des compilateurs certifiés pour des cas d'extrême criticité. Le matériel peut également être vérifié par simulation.

Dans ce document, nous nous intéressons à la vérification des modèles. L'idée est de représenter le système en utilisant un formalisme dédié pour exprimer les caractéristiques concernées par les propriétés à vérifier. On peut alors vérifier que le modèle satisfait les propriétés, en utilisant des techniques formels. Notons qu'au lieu de vérifier de façon passive, les systèmes peuvent aussi être monitorés, c'est-à-dire que les non-conformités peuvent être bloquées à la volée. On peut également forcer le système à respecter une spécification en modifiant certains comportements à l'aide d'un contrôleur ou en forçant la propriété durant l'exécution en stockant les sorties dans un tampon jusqu'à être sûr que la propriété est satisfaite.

Automates temporisés et autres modèles pour les systèmes temps-réel Nous nous intéressons à la vérification de modèles de systèmes temps-réel. Différents formalismes peuvent être utilisés pour représenter ces systèmes. Tout d'abord, un modèle bien connu est celui des réseaux de Petri temporels [Mer74], une extension des réseaux de Petri. Rappelons que les réseaux de Petri son composés d'un ensemble de places qui sont connectées par des transitions. Une transition connecte deux ensembles de places et peut être tirée s'il y a suffisamment de jetons dans le premier ensemble (pré-condition) et l'activation de la transition supprime les jetons spécifiés par la pré-condition et ajoute des jetons dans le second ensemble de places (post-condition). Une exécution d'un réseau de Petri commence avec un nombre de jetons donné dans certaine place (c'est ce qu'on appelle le marquage

initial). Les aspects temporisés peuvent être ajoutés de différentes manières. Dans les réseaux de Petri temporels [Mer74], les contraintes de temps sont mises sur les transitions. Quand la pré-condition d'une transition est satisfaite, une horloge est réinitialisée et la transition pourra être tirée lorsque la valeur de l'horloge appartiendra à l'intervalle associé à la transition. Notons qu'il existe d'autres extensions des réseaux de Petri prenant en compte le temps mais elles sont moins utilisées. Par exemple, les réseaux de Petri temporisés dont les transitions ont des durées d'exécution fixées [Ram74]. Les aspects temporels peuvent également concerner les jetons en considérant leurs âges comme dans les réseaux de Petri à arcs temporisés [Han93].

En plus des réseaux de Petri, un autre modèle usuel a été étendu pour représenter des systèmes temps-réel : les diagrammes haut niveau de séquences de messages [IT11]. Les diagrammes de séquences de messages fournissent une représentation simple des scénari de communication entre processus d'un système. Plus précisément, chaque processus envoie et reçoit des messages dans un certain ordre fixé. Cela induit des contraintes sur le comportement des autres processus. En particulier, un message ne peut pas être reçu avant son émission. Néanmoins, il peut y avoir plusieurs manières correctes d'entrelacer ou de linéariser les exécutions locales des processus. Par exemple, si un processus p envoie un a et un b à un processus q, et q les reçoit dans le même ordre, alors le a peut être reçu par q avant ou après l'envoi du b par p. Les diagrammes haut niveau de séquences de messages sont des graphes permettant de définir des diagrammes de séquences de messages par concaténation de motifs. La sémantique d'un chemin correspond à la concaténation des motifs consécutivement rencontrés le long du chemin. Finalement, les diagrammes haut niveau de séquences de message temporellement contraints (ou *TC-MSC graphs*) [IT11] sont une extension de ces graphes où des contraintes temporelles sont ajoutées le long des exécutions des processus. Des intervalles sont ainsi placés entre deux actions d'un processus pour spécifier les délais autorisés entre ces deux actions.

Finalement, les automates temporisés ont été introduit par Rajeev Alur et David L. Dill au début des années 90 [AD90, AD94]. Grossièrement, un automate temporisé est un automate fini équipé d'horloges continues qui évoluent de façon synchrone. Les arcs sont étiquetés avec des gardes sur les horloges (*e.g.* l'horloge x est supérieure à 1) et peuvent réinitialiser des horloges à zéro. La recherche autour de ce modèle est très active. Plusieurs problèmes (comme les problèmes de la déterminisabilité des automates temporisés ou de l'universalité des langages temporisés reconnus par des automates temporisés) sont indécidables, mais la principale raison du succès de ce modèle est l'abstraction des régions qui permet, par exemple, de décider le problème de l'accessibilité des localités en espace polynomial [AD94]. Dans ce document, nous nous intéressons à la vérification des automates temporisés.

Les réseaux de Petri et les diagrammes de séquences de messages modélisent naturellement la concurrence alors que ce n'est pas le cas des automates temporisés. Pour cela, on peut considérer des réseaux d'automates temporisés. L'avantage des automates temporisés est que le problème de l'accessibilité est décidable en espace polynomial alors que c'est indécidable pour les réseaux de Petri temporels [JLL77] et les TC-MSC graphes [GMNK09]. Néanmoins, une abstraction a été développée pour décider le problème de l'accessibilité dans les réseaux de Petri temporel bornés en espace polynomial [BD91]. Malheureusement, le caractère borné des réseaux de Petri temporels est indécidable.

Notons que les systèmes temps-réel peuvent aussi être modélisés avec une grande expressivité par des automates hybrides qui généralisent les automates temporisés en remplaçant les horloges par des fonctions continues dont les valeurs sont décrites par des équations différentielles [ACHH92]. Malheureusement, l'expressivité a un coût et les classes d'automates hybrides pour lesquelles le problème de l'accessibilité est décidable sont très restrictives.

L'abstraction des régions et ses conséquences Les états des automates temporisés sont des couples avec une localité et une valuation d'horloges (*i.e.* une fonction associant une valeur réelle à chaque horloge). Les automates temporisés ont donc un nombre d'états non-dénombrable. L'idée de l'abstraction des régions est que certaines valuations d'horloges sont similaires et peuvent être traitées ensemble, par exemple $(x = 2.3, y = 5)$ et $(x = 2.35, y = 5)$. Plus précisément, les régions sont des classes d'équivalence sur les valuations d'horloges. Deux valuations appartiennent à la même région si elles satisfont les mêmes gardes et si leurs successeurs satisferont les mêmes gardes. En d'autres mots, depuis un état avec l'une ou l'autre des valuations, on peut suivre les mêmes chemins dans l'automate temporisé. Les automates temporisés peuvent alors être quotientés par la relation d'équivalence des régions en un automate fini. Une première application est la décidabilité du problème de l'accessibilité d'une localité de l'automate temporisé en espace polynomial [AD94]. Plus généralement, l'automate des régions permet de décider les propriétés de sûreté, les propriétés ω-régulières ou les propriétés non-temporisées exprimées en logique temporelle (*linear temporal logic* ou LTL) [Pnu77] ou en logique arborescente (*computation tree logic* ou CTL) [CE81].

Limites de l'abstraction finie Les automates temporisés peuvent être vus comme des accepteurs de langages temporisés, simplement en choisissant un ensemble de localités acceptantes, de la même façon que pour les automates finis. Ils peuvent ainsi accepter des mots temporisés, finis ou infinis suivant la sémantique. Certaines transitions peuvent être internes, c'est-à-dire qu'elles n'ajoutent pas de lettres au mot lu. On dit que deux automates temporisés sont équivalents s'ils acceptent le même langage. Notons que les automates temporisés avec transitions internes sont strictement plus expressifs que sans [BGP96]. Cela signifie qu'il existe des automates temporisés avec transitions internes qui n'admettent pas d'automate temporisé équivalent sans transition interne. Décider le problème du vide d'un langage de mots temporisés pour les automates temporisés, c'est exactement décider le problème de l'accessibilité des localités acceptantes (et donc faisable par l'abstraction des régions). The problème du vide pour les langages de mots infinis peut également être décidé en utilisant l'automate des régions.

L'abstraction par un automate fini a tout de même des limites. Le problème de l'universalité est de savoir si un langage contient tous les mots temporisés. D'autre part, le problème de l'inclusion demande si un langage donné est inclus dans un second langage. On peut remarquer que le problème de l'universalité est un cas particulier du problème de l'inclusion où le premier langage est universel. Ces deux problèmes sont indécidables pour les langages représentés par des automates temporisés (finis or infinis) [AD94]. Si les automates temporisés ont au plus une horloge, la preuve de [AD94] n'assure que la récursivité non-primitive. La décidabilité a été un problème ouvert jusqu'en 2004 [OW04]. Plusieurs variantes de ces problèmes ont été étudiées dans [OW04] et [ADOW05] pour tracer la frontière de la décidabilité. Le problème de l'inclusion est, par exemple, décidable pour les mots temporisés finis si l'automate dont le langage est sensé être plus grand, a au plus une horloge et pas de transition interne [OW04]. Le problème de l'universalité est donc aussi décidable pour cette classe d'automates temporisés. En revanche, le problème de l'universalité (et donc le problème de l'inclusion) devient indécidable pour les automates ayant au moins deux horloges, des transitions internes ou si l'on considère les mots temporisés infinis.

Un automate temporisé est complémentable si on peut construire un automate temporisé acceptant le complémentaire du langage. Les problèmes de l'universalité et de l'inclusion sont proches du problème de la complémentabilité. En effet, ils sont décidables pour les automates temporisés complémentables, grâce à la même astuce que pour les automates finis. Les automates temporisés ne sont pas complémentables en général, même s'ils n'ont qu'une horloge [AD94], ce qui rend sur-

prenante la décidabilité du problème de l'inclusion pour les automates avec une seule horloge. D'un autre côté, les automates temporisés déterministes (*i.e.* ayant au plus une exécution lisant chaque mot) sont facilement complémentables, en intervertissant les localités acceptantes et non-acceptantes. Les automates temporisés ne sont donc pas déterminisables en général [AD94].

0.2 Déterminisation des automates temporisés et application au test

Déterminisation des automates temporisés La déterminisation est utile dans beaucoup de contextes quel que soit le modèle utilisé. C'est une clé pour plusieurs problèmes tels que l'implémentabilité des modèles, parce qu'une implémentation est nécessairement déterministe. C'est aussi intéressant pour le diagnostic de faute, où l'on veut détecter si une trace d'un système mène nécessairement à un évènement fautif, c'est-à-dire si pour toute exécution correspondant à cette trace, la localité courante est fautive. Pour la génération de cas de test, la déterminisation permet de connaître toutes les sorties autorisées après une trace, dans le but d'émettre un verdict correct. Plus généralement, la déterminisation est utile pour les problèmes où l'analyse sous-jacente dépend du comportement observable ou pour les problèmes où une complémentation du modèle est nécessaire.

L'impossibilité de déterminiser les automates temporisés peut alors être gênante. De plus, le problème de la déterminisabilité des automates temporisés est indécidable, même si l'on fixe le nombre d'horloges et la constante maximale [Tri06, Fin06]. Des solutions partielles sont donc apportées.

Suivant le contexte, deux approches sont possibles quand on a besoin de déterminiser des automates temporisés: soit on se restreint à des classes d'automates déterminisables, soit on fait une déterminisation approchée. Ces deux pistes ont été récemment explorées.

Premièrement, il existe une procédure de déterminisation [BBBB09] qui fournit un automate temporisé déterministe équivalent à l'automate temporisé original, mais elle ne termine pas pour tous les automates temporisés. Néanmoins, cette procédure termine pour tous les automates des classes connues d'automates temporisés déterminisables (les automates temporisés fortement non-Zeno [AMPS98], les automates temporisés *event-clock* [AFH94] ou encore les automates temporisés à réinitialisations entières [SPKM08]).

Deuxièmement, un algorithme de déterminisation sur-approchée [KT09] a été développé pour construire un automate temporisé déterministe acceptant au moins les mots acceptés par l'automate temporisé original. Cette approche utilise une politique de réinitialisation fixée *a priori* pour les horloges de l'automate temporisé déterministe. On essaie ensuite de simuler le comportement de l'automate temporisé original en utilisant des estimées d'états. Cette approche ne préserve pas le langage, même pour des automates déterministes. En revanche, elle termine toujours.

Contribution 1 : une approche par le jeu pour déterminiser les automates temporisés Dans cette thèse, nous proposons une combinaison des deux approches. Nous utilisons des estimées d'états similaires à celles de [KT09]. Le principe de la méthode est de construire un jeu pour trouver une politique de réinitialisation adaptée pour minimiser le risque d'approximation. Notre algorithme est plus précis que la sur-approximation existante [KT09] et de plus, il déterminise de façon exacte, strictement plus d'automates temporisés que l'autre approche existante [BBBB09]. Cette contribution a été publiée dans le papier [BSJK11a].

Application de la déterminisation au test basé sur les modèles Dans ce document, nous développons l'application de la déterminisation à la sélection de tests. Dans la littérature, on trouve plusieurs problèmes de test. Nous nous concentrons sur le test de conformité de modèles. Étant donnée une

spécification et une implémentation inconnue, le but est de vérifier que l'implémentation est conforme à la spécification. Le modèle de l'implémentation n'étant pas disponible, on ne peut qu'interagir avec elle en la stimulant par des entrées et en observant les sorties tout en vérifiant leur conformité. Pour cela, on génère des automates temporisés appelés cas de test.

L'objectif de l'exécution des cas de test est de détecter les non-conformités et ainsi d'émettre un verdict d'échec. En effet, on ne peut pas espérer prouver que l'implémentation n'a aucune non-conformité simplement par le test. La propriété principale pour les cas de test, appelée correction, est donc le fait que tout verdict d'échec correspond à une non-conformité. Pour émettre des verdicts corrects, on a besoin de connaître toutes les sorties autorisées par la spécification après l'observation courante. On a donc besoin de calculer l'ensemble des états accessibles après cette observation. Ceci est très proche de la déterminisation. En fait, soit la spécification est déterministe (ou déterminisée hors-ligne), soit on doit, en quelque sorte, la déterminiser à la volée (dans le cas du test en-ligne).

Contribution 2 : génération hors-ligne de cas de test Dans cette thèse, nous proposons une approche formelle pour la génération hors-ligne de cas de tests pour les automates temporisés non-déterministes. Nous utilisons des objectifs de test modélisés par des automates temporisés qui sont capables d'observer les horloges de la spécification. Leur rôle est de guider la génération des cas de test. Notre modèle intègre l'observabilité partielle et l'urgence qui sont primordiales pour modéliser des systèmes réactifs réalistes. Comme les automates temporisés ne sont pas déterminisables, la plupart des contributions sur le test d'automates temporisés sont restreintes à des spécifications déterministes ou déterminisables [KJM04, NS03]. Une exception notable est [KT09] qui gère le problème en utilisant l'algorithme de déterminisation approchée dont nous avons discuté plus tôt. Leur modèle intègre l'urgence, malheureusement la déterminisation approchée la supprime complètement. Dans ce cas, l'implémentation qui ne fait rien est conforme à toutes les spécifications. Au contraire, les extensions de notre approche par le jeu pour la déterminisation des automates temporisés permettent de traiter les transitions internes qui modélisent l'observabilité partielle, et les invariants qui modélisent l'urgence. De plus, la construction de notre jeu est adaptée pour traiter différemment les entrées et les sorties. Ceci permet de préserver la relation de conformité **tioco** [KT09] sans restrictions sur les spécifications, contrairement à [KT09]. Ce travail a été publié dans [BJSK11, BJSK12].

0.3 Fréquences dans les automates temporisés

Aspects quantitatifs dans les automates temporisés Dans ce document, nous nous intéressons à l'ajout d'aspect quantitatifs dans la vérification des automates temporisés. Notre but est, par exemple, de modéliser des problèmes de consommations d'énergie, de ratios d'erreur ou de risques d'erreur. Récemment, plusieurs approches ont été proposées dans cet esprit. Les notions de volume et d'entropie ont été introduites pour les langages temporisés lus par des automates temporisés et plusieurs moyens de les calculer ont été proposés [AD09a, AD09b, AD10]. D'un autre côté, une sémantique probabiliste pour les automates temporisés a été introduite [BBB$^+$08]. Cette sémantique résout le non-déterminisme avec une sorte d'équité, grâce à une distribution de probabilité. La vérification quantitative de propriétés ω-régulières peut alors être faite pour les automates temporisés à une horloge [BBBM08]. En d'autres termes, on sait décider si la probabilité de satisfaire une propriété ω-régulière donnée, satisfait une contrainte de seuil pour un seuil rationnel. La méthode utilise un algorithme permettant de calculer la probabilité de satisfaire la propriété si le seuil est rationnel, et qui l'approche à une précision donnée sinon. Notons que d'autres modèles combinent des aspects temporisés et probabilistes. Par exemple, dans [KNSS02] une variante des automates temporisés dont les

horloges peuvent être mises à jour avec une valeur aléatoire a été enrichie en ajoutant des transitions dont la cible est aléatoire. Une méthode approchée (avec estimation d'erreur) a alors été proposée pour le *model-checking* de propriétés quantitatives telles que "la probabilité maximale de passer infiniment souvent par une localité donnée est plus grande que $\frac{1}{3}$". Finalement, des travaux traitent des automates temporisés avec des coûts ou des doubles prix [ATP01, BFH$^+$01, BBL08]. Des coûts ou des couples coût-récompense sont définis sur les arcs et les localités. Une valeur est ainsi associée à chaque exécution en prenant en compte les coûts rencontrés le long de l'exécution (min, max, moyenne...). Il est alors possible, par exemple, de vérifier des propriétés telles que "le coût minimal pour accéder à une localité donnée est plus petit que 3" [ATP01, BFH$^+$01]. On sait également calculer un ordonnancement optimal dans un automate temporisé avec coûts et récompenses [BBL08].

Contribution 3 : automates temporisés avec fréquences Pour les mots temporisés infinis, la condition d'acceptation habituelle est la sémantique de Büchi, c'est-à-dire qu'une exécution est acceptée si elle passe infiniment souvent par des localités acceptantes. Cette sémantique n'est pas adaptée à tous les contextes. Par exemple, si les localités acceptantes modélisent les états du système dans lesquels des ressources sont utilisées de façon optimale, ou dans lesquels certaines actions sont particulièrement bon marché, le temps écoulé dans ces localités acceptantes peut être important. Dans cette thèse, nous introduisons la notion de fréquence d'une exécution comme étant la proportion de temps écoulé dans les localités acceptantes. Nous définissons alors des sémantiques avec des conditions d'acceptation prenant en compte la fréquence. Par exemple, une exécution peut être acceptée si sa fréquence est plus grande que $\frac{2}{3}$. Cela permet de définir des langages temporisés prenant en compte des aspects quantitatifs. Ensuite, nous présentons des techniques pour calculer les bornes de l'ensemble des fréquences des exécutions d'un automate temporisé, dans le but de décider les problèmes du vide et de l'universalité de ces langages. La méthode est basée sur un raffinement de l'abstraction des régions, appelé abstraction des coins [BBL08]. Ceci permet d'abstraire le temps écoulé le long des exécutions, et ainsi de définir une fréquence abstraite en utilisant des coûts et des récompenses. Dans un premier temps, nous ne considérons que les automates temporisés à une horloge, car leur comportement temporel est plus simple. Cette contribution a été publiée dans le papier [BBBS11].

Réalisabilité et convergences D'un point de vue implémentabilité, les automates temporisés ne sont pas toujours réalistes. En effet, la précision absolue des horloges est irréaliste. Plusieurs papiers à propos de la robustesse des comportements des automates temporisés ont été récemment publiés (voir [Mar11] pour un *survey*). Intuitivement, les gardes sont légèrement modifiées et les horloges peuvent dériver légèrement pour prendre en compte les imprécisions possibles, on vérifie alors si certaines propriétés sont préservées. D'un autre côté, pour être en mesure d'observer certains phénomènes de convergence le long des exécutions infinies, il serait nécessaire de disposer d'horloges de plus en plus précises au fil de l'exécution. Le plus connu est le phénomène Zeno, où le délai accumulé le long d'une exécution infinie est borné. Avec deux horloges, d'autres types de convergences peuvent apparaître. En effet, le long d'un cycle à deux horloges, il est possible de forcer les délais écoulés dans une localité donnée à décroîtrent. D'autre part, dans le contexte du calcul de l'entropie des langages d'automates temporisés, la notion de cycle amnésique a été définie pour caractériser les cycles n'exhibant pas de telles convergences. Un moyen de décider si un cycle est amnésique est présenté dans [BA11]. Comme les convergences ne sont pas réalistes, il est raisonnable de supposer que le système n'admet pas de tels phénomènes, tout en gardant à l'esprit qu'il serait intéressant de savoir décider si un automate temporisé satisfait l'hypothèse d'amnésie.

Contribution 4 : fréquences dans les automates temporisés amnésiques Les techniques utilisées pour les automates temporisés à une horloge ne peuvent pas s'appliquer aux automates temporisés à plusieurs horloges. De plus, toutes les illustrations de cette limite contiennent des phénomènes de convergence forcée le long de cycles. Dans cette thèse, nous proposons donc une extension de l'étude de l'ensemble des fréquences pour les automates temporisés à plusieurs horloges, en supposant qu'ils n'admettent pas de phénomènes de convergence (ou sous des hypothèses plus faibles donc la satisfaction est décidable). Le calcul est toujours effectué grâce à l'abstraction des coins. L'ensemble des fréquences abstraites est calculé dans l'abstraction. Ensuite, nous prouvons que cet ensemble est égal à l'ensemble des fréquences dans l'automate temporisé, en utilisant des propriétés des cycles amnésiques. Cette extension a été publiée dans le papier [Sta12].

0.4 Accessibilité dans les automates temporisés communicants

Communication entre automates temporisés Nous considérons maintenant des systèmes temps réel distribués où les processus peuvent communiquer. Un premier défi est de les modéliser avec des formalismes adaptés. Pour cela, il faut trouver le bon compromis entre l'expressivité et la complexité. Malheureusement, le problème de l'accessibilité est indécidable dans la plupart des modèles possibles, même sans aspects temporels. Le second défi est donc de trouver des restrictions, les plus faibles possible, qui rendent le problème de l'accessibilité décidable.

Des systèmes composés de plusieurs automates temporisés peuvent être considérés avec différentes modélisations de la communication, de la même façon que pour les automates finis. Ils peuvent principalement communiquer par synchronisation sur des actions (*e.g.* input/ouput), par des canaux (parfaits ou imparfaits) ou encore par *broadcast*. Pour les automates temporisés, un autre moyen de communication utilisant les valeurs des horloges a été exploré [ABG$^+$08]. Les résultats sont assez surprenant. Les automates temporisés distribués ont des processus avec des horloges locales qui évoluent à des vitesses indépendantes. Chaque automate temporisé observe les horloges des autres sans pouvoir les réinitialiser. Dans un tel modèle, on ne peut pas décider si un mot non-temporisé donné peut être lu quelque soit les décalages d'horloges, même si les différences ou les ratios d'horloges sont bornés.

Les machines communicantes à états finis (*i.e.* les automates finis communiquants via des canaux non-bornés) [Pac82] sont un modèle fondamental pour les systèmes distribués. Une extension naturelle de ce modèle est de considérer des automates temporisés au lieu des automates finis. Ce modèle a été étudié dans [KY06]. Pour les machines communicantes à états finis, il est bien connu que le problème de l'accessibilité est décidable si et seulement si la topologie est une polyforêt (*i.e.* un graphe sans cycle indirect) [Pac82, BZ83]. Ajouter des aspects temporels dans ce modèle mène à un résultat d'indécidabilité très fort [KY06]. Plus précisément, le problème de l'accessibilité a été prouvé indécidable pour les *pipelines* (*i.e.* suites de processus où chaque processus peut envoyer des messages au suivant) si et seulement s'il y a au moins trois processus dans le pipeline.

Contribution 5 : accessibilité dans les automates temporisés communicants Dans [KY06], la preuve d'indécidabilité utilise une réduction du problème de l'accessibilité dans les machines à deux compteurs avec tests à zéro. Or on remarque que les tests à zéro sont simulés dans le pipeline d'automates temporisés grâce à l'urgence des réceptions (*i.e.* les réceptions sont prioritaires sur les actions internes). Nous avons décidé d'étudier la décidabilité du problème de l'accessibilité en supprimant l'hypothèse d'urgence sur les canaux. Pour cela, nous étudions d'abord plus un modèle plus simple d'automates communicants tic-tac, c'est-à-dire des automates finis communiquants

et se synchronisant sur une action discrète. Ce modèle peut être vu comme un système de processus communicants à temps discret. Ensuite, nous réduisons le problème de l'accessibilité dans les processus communicants à temps continu au problème de l'accessibilité dans les processus communicants à temps discret. De cette façon, sans urgence, les topologies pour lesquelles le problème de l'accessibilité est décidable sont les mêmes pour les machines communicantes à états finis, les automates communicants tic-tac et les automates temporisés communicants. Nous proposons, de plus, une caractérisation plus précise des topologies décidables pour les systèmes de processus temporisés communiquants par canaux urgents et non-urgents. Ce travail a été publié dans le papier [CHSS13].

0.5 Conclusion

Nous avons contribué dans trois différentes branches de la vérification des automates temporisés, à savoir la déterminisation des automates temporisés, la vérification d'automates temporisés avec des aspects quantitatifs et la vérification d'automates temporisés communicants. La suite du manuscrit détaille largement ces contributions ainsi que l'état-de-l'art de ces trois directions. La langue employée est l'anglais pour pouvoir être lue par un jury de thèse international.

Part II

Introduction

Chapter 1

Introduction

In daily life as well as in very specialized contexts, we meet more and more systems governed by software. They play an increasingly important role in our society and we need to be sure that they behave as expected. Beyond reacting in a correct manner in any situation, most of these systems must satisfy explicit response-time constraints or risk severe consequences, including failure. These are called real time systems. The correctness of such systems is based on the correctness of the outputs but also of their delays. Their clocks can have very different precisions depending on the context. As a first example, real time systems are present in the aerospace industry. In avionic and space transportation systems, computers are used to monitor and control plane and space shuttle missions. On the other hand, in planetary exploration applications, real time systems collect and analyze data from space exploration missions. In these situations, systems have to be reactive already in the moving management. In such contexts, failures can cause human losses or have expensive costs. Real time systems are also very useful for production processes in industry. The production of high commodity chemicals such as ethylene and prophylene is supervised and controlled by computers. These systems require high performance real-time features and should provide interfaces to regulatory control instrumentation systems. Any failure here may cause an environmental disaster.

Verification of real time systems The list of critical applications of real time systems is very large and yields as many reasons to work on their verification. Expected properties for a given real time system can be represented in several ways. Depending on the criticality level, on the knowledge about the system and on the properties which have to be checked, a lot of approaches are possible. If the system is a black box, test cases can be generated in order to communicate with the system and detect non-conformances with respect to its specification. Otherwise, the verification can be done at several levels of abstraction. Programs can be proved, with even the certification of compilers for extreme criticality. Moreover, hardware can also be checked by performing simulations. In this document, we mainly focus on model-based verification. The idea is to represent the system using a dedicated formalism expressing features concerned by the considered properties. One can then check that the model satisfies the properties using formal methods. On the other hand, instead of a passive verification, systems can also be monitored, that is non-conformances can be blocked, or one can force systems to respect a specification by restricting behaviors with a controller, or enforcing the property during the execution by buffering outputs until one is sure that the property is satisfied.

Timed automata and other models for real time systems We are interested in the model-based verification of real time systems. Different formalisms may be used to model these systems. A first

well-known model is time Petri nets [Mer74] an extension of Petri nets. Recall that Petri nets are composed of a set of places which are connected by transitions. A transition connects two sets of places, and can be fired if there are enough tokens in the first set (pre-condition) and the activation of the transition removes the tokens of the pre-condition to add some tokens in the second set (post-condition). An execution of a Petri net starts with some tokens in some places (initial marking). Timing aspects can then be added in several ways. In time Petri nets [Mer74], timing constraints are put on transitions. When the pre-condition of a transition is satisfied, a clock is reset, and the transition can be fired when the clock value belongs to the interval associated with the transition. Note that there are other time extensions of Petri nets which are less used. A first example is timed Petri nets whose transitions have a fixed duration [Ram74]. Timing behaviors can also be supported by tokens considering the ages of the tokens as in timed-arcs Petri nets [Han93].

Beyond Petri nets, another famous model has been extended to represent real time systems: high-level message sequence charts [IT11]. Message sequence charts are a simple way to represent communication scenarios between processes of a system. More precisely, each process sends and receives messages in a certain order which is fixed. This may induce some constraints over the behaviors of the other process. In particular a message cannot be received before its emission. Nevertheless, there can be several consistent interleavings or linearizations of the local executions. For example if a process p sends a and b to a process q, and q receives them in the same order, then a can be received by q before or after the emission of b by p. High-level message sequence charts (HSMCs) are graphs allowing to define message sequence charts by concatenation of given basic message sequence charts. The semantics of a path thus corresponds to the concatenation of the patterns which are consecutively visited. Finally, time-constrained message sequence charts graphs (TC-MSC graphs) [IT11] are an extension of HSMCs where some timing constraints are put along the local runs of the processes as intervals of allowed values for delays between two actions of a given process.

Finally, timed automata have been introduced by Rajeev Alur and David L. Dill in 1990 [AD90, AD94]. A timed automaton is roughly a finite automaton equipped with continuous clocks which evolve in a synchronous way. Edges are labeled with guards over these clocks (*e.g.* clock x is larger than 1) and can reset them to zero. Research around this model is very active. Several verification problems are undecidable, but the main cause of the celebrity of timed automata is the region abstraction which allows one to decide reachability of locations in polynomial space [AD94]. In this document, we are interested in the verification of timed automata.

Time Petri nets and TC-MSC graphs naturally model concurrency, whereas timed automata do not, but it is possible to consider nets of timed automata. The advantage of timed automata is that reachability can be decided in polynomial space [AD94], whereas it is undecidable for time Petri nets [JLL77] and TC-MSC graphs [GMNK09]. Nevertheless, a nice abstraction has been developed to decide reachability in bounded time Petri nets in polynomial space [BD91], unfortunately the boundedness of time Petri nets is undecidable.

Note that real time systems can also be modeled with a great expressivity by hybrid automata which generalize timed automata, using continuous variables whose values are described by ordinary differential equations [ACHH92]. Unfortunately, expressivity has a cost, and classes of hybrid automata for which the reachability problem is decidable are very restrictive.

The region abstraction and consequences States of timed automata are couples with a location and a valuation of clocks (*i.e.* a function assigning a real value to each clock). Timed automata thus have an uncountable state space. The idea of the region abstraction [AD90, AD94] is that some valuations of clocks are very similar and can be considered together, *e.g.* $(x = 2.3, y = 5)$ and

$(x = 2.35, y = 5)$. Regions are equivalence classes over the valuations of clocks. Two valuations belong to the same region if they satisfy the same guards, and their successors will satisfy the same guards in the future. In other words, from a state with one or the other valuation, the same paths can be followed in the timed automaton. Timed automata can then be quotiented by the region equivalence relation into a finite automaton (called the region automaton). The region equivalence relation is then a time-abstract bisimulation between both automata. This roughly means that the untimed behaviors are preserved. For example, untimed words which are accepted by a timed automaton with some timestamps are accepted in the region automaton. A first application of the region abstraction is the decidability of the reachability of locations in timed automata in polynomial space. More generally, the region automaton allows to decide all the properties preserved by time-abstract bisimulation. As a consequence, one can decide safety properties, ω-regular properties, or untimed properties expressed in linear temporal logic [Pnu77] or in computational tree logic [CE81].

Limits of the finite abstraction Timed automata can be seen as timed languages acceptors simply choosing a set of accepting locations in the same way as for finite automata. They can thus accept finite or infinite timed words depending of the semantics. Some transitions can be internal, that is they do not add letters to the word. We say that two timed automata are equivalent if they accept the same language. Note that timed automata with internal transitions are strictly more expressive [BGP96] than timed automata, that is, there exist timed automata with internal actions for which we cannot build an equivalent timed automaton without internal action. Then, the decision of the emptiness of the language of finite timed words of a timed automaton is exactly the decision of the reachability of the accepting locations (feasible thanks to the region automaton). The emptiness for languages of infinite timed words can also be decided in the region automaton.

Nevertheless, the abstraction by a finite automaton naturally has some limits. The universality problem asks whether a language contains all timed words. Moreover, the inclusion problem asks, given two languages, whether the first one contains all the words of the second one. Remark that the universality problem is a particular case of the inclusion problem where the first language is universal. Both problems are undecidable [AD94] for languages defined by timed automata (for finite or infinite words). If timed automata have at most one clock, the hardness proof of [AD94] only ensures the non-primitive recursivity, and the decidability has been an open question until 2004 [OW04]. Several variants of these problems have been studied in [OW04] and [ADOW05] to draw the frontier of undecidability. The inclusion problem is, for example, decidable for finite timed words if the timed automaton whose language is checked to be larger, has at most one clock and no internal transition [OW04]. As a consequence, the universality problem is also decidable for this class of timed automata. However, the universality problem (and hence the inclusion problem) becomes undecidable if timed automata have two clocks, internal actions, or if we consider infinite timed words [ADOW05].

The universality problem and the inclusion problem are close to the complementability problem, because for complementable timed automata, they become decidable with the same trick as for finite automata. Some timed automata cannot be complemented, even when only one clock is allowed [AD94], which makes the decidability result about the inclusion problem quite surprising. On the other hand, deterministic timed automata (*i.e.* having at most one run reading each timed word) are easily complementable, by switching accepting and non-accepting locations. As a consequence, timed automata are not determinizable in general [AD94].

Determinization of timed automata and application to testing

Determinization of timed automata Determinization is useful in a lot of contexts whatever the model considered. It is a key issue for several problems such as implementability of models, because an implementation is necessarily deterministic. It is also interesting for fault diagnosis, where we want to detect if a trace of the system surely leads to a faulty location, that is for all the runs corresponding to this trace, the current location is a fault location. For test generation, determinization allows one to foresee the set of allowed outputs, in order to emit a sound verdict. More generally, determinization is helpful for problems where the underlying analyses depend on the observable behavior or where complementation is required.

Then, the unfeasibility of the determinization of timed automata can be embarrassing. Moreover, the determinizability of timed automata, even when fixing the number of available clocks and the maximal constant, is undecidable [Tri06, Fin06]. Then, partial solutions to this problem have been proposed. Depending on the context, two approaches are possible when it is necessary to determinize timed automata: either restricting to classes of determinizable timed automata, or performing an approximate determinization. Both approaches have been recently explored.

First, there exists a determinization procedure [BBBB09] which yields a deterministic timed automaton equivalent to the argument timed automaton, but does not terminate for all timed automata. Nevertheless, it terminates for all the known determinizable classes: strongly non-Zeno timed automata [AMPS98], event-clock timed automata [AFH94] or timed automata with integer resets [SPKM08].

On the other hand, an approximate determinization [KT09] has been developed to build a deterministic timed automaton whose language contains at least the words accepted by the argument timed automaton. This method *a priori* fixes a reset policy for the clocks of the deterministic timed automaton, and then tries to simulate behaviors of the argument timed automaton using state estimates. It does not necessarily preserve languages, even for deterministic timed automata, but always terminates.

Contribution 1: determinization of timed automata In Chapter 3, we propose a combination of both approaches. We use a state estimate which is similar to the one of [KT09]. However, rather than fixing a reset policy, the principle of our method is to build a game in order to find a suitable reset policy that tries to avoid to approximate. Our algorithm is more precise than the existing over-approximation [KT09], and moreover also yields an exact determinization for strictly more timed automata than the other existing approach [BBBB09]. This contribution has been published in the paper [BSJK11a].

Application of the determinization to model-based testing In this document, we develop the application to test selection. In the literature, there exist several testing problems but we focus on the model-based conformance testing. Given a specification and an unknown implementation, we aim at checking whether the implementation conforms to the specification. The model is not available, and one can only interact with the implementation by sending inputs and observing outputs and checking their conformance with respect to expected ones. To do so, test cases are generated as determinisitic timed automata.

The goal of test execution is to detect non-conformances and emit a fail verdict in this case. One cannot ensure that the implementation has no non-conformance only by testing. The main property that is requested for test cases, called soundness, is thus that fail verdicts always correspond to non-

conformances. In order to emit a sound verdict, one needs to foresee the outputs allowed by the specification after the current observation. Then, it requires to compute the set of states reachable after this observation. This is very close to determinization. In fact, either the specification is deterministic (or determinized off-line), or it has to be determinized on-the-fly (in the case of on-line testing). The main drawback of this latter possibility is the computation time. In the context of real time systems, the on-the-fly computation could hinder the testing process by taking too much time.

Contribution 2: off-line test selection In Chapter 4, we propose a general formal framework for the off-line test selection for non-deterministic timed automata. We use test purposes modeled by timed automata which are able to observe clocks of the specification to finely guide the generation of test cases. Moreover, our model integrates partial observability and urgency. Because of the non-determinizability of timed automata, most of contributions about testing of timed automata are restricted to deterministic or determinizable specifications [KJM04, NS03]. A notable exception is [KT09] where the problem is solved by the use of the over-approximate determinization discussed above. Their model contains urgency, unfortunately the approximate determinization removes it. In this case, an implementation which does nothing, conforms to all specifications. On the contrary, extensions of our game approach for the determinization allow to deal with ε-transitions modeling partial observability, and invariants modeling urgency. Moreover, the construction of the game is adapted to preserve the conformance relation **tioco**, without assumptions over the specification contrary to [KT09]. This work has been published in [BJSK11, BJSK12].

Frequencies in timed automata

Quantitative aspects In this Part IV, we are interested in adding quantitative aspects in timed automata verification. For example to model problems about energy consumptions, failure rates or risks of failure. Recently, several approaches have been proposed in the same spirit. Volumes and entropies of timed languages recognized by timed automata have been defined and several ways to compute them have been studied [AD09a, AD09b, AD10]. On the other hand, a probabilistic semantics for timed automata has been introduced [BBB$^+$08]. The non-determinism both over delays and over enabled moves in the execution of a timed automaton is treated with a kind of fairness assumption thanks to a probability distribution. Then, the quantitative model checking of ω-regular properties in one-clock timed automata can be performed [BBBM08]. In other words, one can decide whether the probability to satisfy an ω-regular property, satisfies a threshold condition for a rational threshold. The approach is based on an algorithm which allows to compute the probability to satisfy the property if it is a rational number, and to approximate it up to an arbitrary precision otherwise. Note that other models combining time and probabilities have been defined. For example, in [KNSS02] a variant of timed automata whose clocks can be set with random values is enriched, defining transitions whose target is random. For one-clock timed automata, an approximate approach (with estimation of the error) has then been proposed for the model checking of quantitative properties such as "the maximum probability to visit infinitely often a given location is larger than $\frac{1}{3}$". Finally, several recent works concern timed automata equipped with costs or double prices [ATP01, BFH$^+$01, BBL08]. Costs or cost-reward pairs are associated with edges and locations and a value is assigned to each run taking into account the costs along the run It is, for example, possible to check properties such as "the minimal cost to reach a given location is smaller than 3", or to compute an optimal infinite scheduling in a double-priced timed automaton.

Contribution 3: frequencies in timed automata For infinite timed words, the usual acceptance condition in timed automata is the Büchi semantics, that is, a run is accepting if it visits infinitely often accepting locations. This semantics is not always suitable. For example, if accepting locations model states of a system in which resources are optimally used, or in which some actions are particularly cheap, the time elapsed in accepting locations can be important. In Part IV, we introduce the notion of frequency of a run as the proportion of time elapsed in accepting locations along the run. We thus define semantics with frequency-based acceptance conditions. For instance, a run can be accepted if its frequency is larger than two thirds. This allows to define timed languages taking into account quantitative aspects. Then, we present techniques to compute the bounds of the sets of frequencies of runs in a timed automaton, with the aim to decide emptiness or universality of quantitative languages. The method is based on a refinement of the region abstraction, called the corner-point abstraction [BBL08], allowing to abstract time elapsed along runs, and thus to define an abstract frequency using costs and rewards. In a first phase, we only consider one-clock timed automata whose timed behaviors are simpler. This contribution has been published in the paper [BBBS11].

Realizability and convergences From an implementability point of view, timed automata are not always realistic. Indeed, the absolute precision of clocks is unrealistic. As a consequence, several papers about the robustness of behaviors in timed automata have been recently published (see [Mar11] for a survey). Roughly, guards are lightly modified and clocks can derive a bit, to take into account this lack of precision and one checks whether some properties are preserved. On the other hand, some convergence phenomena along infinite runs may require more and more precise clocks along the run to be observed. The most famous is zenoness. An infinite run is said Zeno if the accumulated delay along the run is finite. With two clocks, other convergences may appear. Indeed, it is possible to force, along a cycle with two clocks, delays in a fixed location to be smaller and smaller. Moreover, in the context of the computation of the entropy of the language of a timed automaton, forgetful cycles have been defined as cycles without such convergences, and a way to decide whether a cycle is forgetful has been presented [BA11]. As convergences are unrealistic, it is reasonable to assume that the system has no such phenomenon, keeping in mind that it would be nice to be able to decide whether a timed automaton satisfies the assumption.

Contribution 4: frequencies in forgetful timed automata Techniques used in the context of frequencies, for one-clock timed automata do not extend to several clocks. Moreover, all the illustrations of this limitation are based on convergence phenomena forced along non-forgetful cycles. We then propose an extension of the study of the set of frequencies to timed automata with several clocks, assuming that they do not admit convergence phenomena (or under weaker assumptions concerning convergences, whose satisfaction is decidable). The computation is again based on the corner-point abstraction. The set of abstract frequencies is computed in the corner-point abstraction, and then proved to be equal to the set of frequencies in the timed automaton, using properties of forgetful cycles. This extension has been published in the paper [Sta12].

Reachability in communicating timed automata

Communication between timed automata We are interested in distributed real-time systems where processes can communicate. A first challenge is to model them with suitable formalisms, finding the good tradeoff between expressivity and complexity. Unfortunately, the reachability problem in most

of the possible models is undecidable, even without timing aspects. The second challenge is thus to find restrictions in which the reachability problem is decidable.

Systems composed of several timed automata can be considered together with several communication policies as for finite automata. They can mainly communicate thanks to synchronization of actions (*e.g.* input/output), by channels (perfect or lossy) or by broadcast. For timed automata, another dimension of communication has been recently investigated [ABG$^+$08] and results can be surprising. Distributed timed automata with local times evolving with their own rates are introduced. Each timed automaton communicates by only observing clocks of the others without the ability to reset them. In such a model, one cannot decide whether a fixed untimed word can be read for all clock drifts, even if the differences, or the ratios of the clock drifts are bounded.

Communicating finite-state machines (*i.e.* finite automata communicating via unbounded channels) [vB78] is a fundamental model for distributed systems. A natural extension of this model, considering timed automata instead of finite automata has naturally been investigated in [KY06]. For communicating finite-state machines, it is well-known that reachability is decidable if and only if the topology is a polyforest [Pac82, BZ83] (*i.e.* without undirected cycle). Adding timing aspects in the model can lead to a stronger undecidability result [KY06]. More precisely, reachability has been shown undecidable for pipelines (*i.e.* sequences of processes where each process can send messages to the following one) if and only if there are at least three automata in the pipeline.

Contribution 5: reachability in communicating timed automata In [KY06], the undecidability proof is based on a reduction of the reachability in a two-counter machine with zero tests. Noting that, zero tests are simulated thanks to urgency of receptions (*i.e.* receptions have priority over internal actions), we decided to explore the decidability of reachability, removing the urgency assumption over channels. To do so, we first study communicating tick automata, that is communicating finite automata synchronizing on a discrete action. This model can be seen as communicating discrete time processes. We then reduce the reachability problem in communicating continuous time processes to the reachability problem in communicating discrete time processes. In this way, we obtain the same decidable topologies for communicating timed automata and communicating tick automata, when channels are not urgent, as for communicating finite automata. Moreover, we present extended characterizations of the decidable topologies considering communicating timed automata with urgent and non urgent channels. This work has been published in the paper [CHSS13].

Outline

In Part III, we present our game approach for the determinization of timed automata and its application to off-line test selection. Then Part IV is devoted to timed automata with frequencies and, in particular, to the study of the set of frequencies in timed automata. Finally, in Part V, we characterize topologies for which the reachability problem is decidable in communicating timed processes.

Chapter 2

Technical preliminaries

In this chapter, we introduce all the fundamental notions used in the sequel of the document. We start by defining valuations of a set of clocks together with some operations on them. Then we present a general definition of timed automata, the syntax and the semantics for finite and infinite runs. Finally, the usual region abstraction is defined in the last section.

Let us define valuations of clocks and operations such as reset and projection. Given a finite set of clocks X, a clock *valuation* is a mapping $v : X \to \mathbb{R}_+$, where $\mathbb{R}_+$ denotes the set of non-negative reals. We note $\overline{0}_X$ the valuation that assigns 0 to all clocks ($\overline{0}$ if X is clear). If v is a valuation over X and $t \in \mathbb{R}_+$, then $v + t$ denotes the valuation which assigns to every clock $x \in X$ the value $v(x) + t$. Moreover, $\overrightarrow{v}$ is the set of the valuations reachable from v letting time elapse, formally $\{v + t \mid t \in \mathbb{R}_+\}$. For $X' \subseteq X$ we write $v_{[X' \leftarrow 0]}$ for the valuation equal to v on $X \setminus X'$ and to $\overline{0}$ on X', $v_{|X'}$ for the valuation v restricted to X', and $v_{[X' \leftarrow 0]^{-1}}$ for the set of valuations v' such that $v'_{[X \leftarrow 0]} = v$.

Guards and invariants are defined as follows. Given a non-negative integer M, an *M-bounded guard* over X, or simply *guard* when M is clear from context, is a finite conjunction of constraints of the form $x \sim c$ where $x \in X,\ c \in [0, M] \cap \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. We denote by $G_M(X)$ the set of M-bounded guards over X.

Given a guard g and a valuation v, we write $v \models g$ if v satisfies g. Formally we define inductively the satisfaction: if $g = (x \sim c)$ with $x \in X,\ c \in [0, M] \cap \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$, then $v \models g$ if $v(x) \sim c$; and if $g = g_1 \wedge g_2$ then $v \models g$ if $v \models g_1$ and $v \models g_2$.

Invariants are restricted cases of guards: given $M \in \mathbb{N}$, an *M-bounded invariant* over X is a finite conjunction of constraints of the form $x \lhd c$ where $x \in X,\ c \in [0, M] \cap \mathbb{N}$ and $\lhd \in \{<, \leq\}$. We denote by $I_M(X)$ the set of invariants.

In the sequel, we sometimes abuse notations by writing in the same way guards or invariants and the sets of valuations which satisfy them.

We are now able to give the definition of timed automata introduced in [AD90, AD94].

Definition 2.1 (Timed automata)**.** *A* timed automaton *(TA) is a tuple* $\mathcal{A} = (L, L_0, F, \Sigma, X, M, E, \mathsf{Inv})$ *such that:* L *is a finite set of* locations*,* $L_0 \in L$ *is the set of* initial locations*,* $F \subseteq L$ *is the set of* final locations*,* Σ *is a finite alphabet,* X *is a finite set of clocks,* $M \in \mathbb{N}$ *the* maximal constant*,* $E \subseteq L \times G_M(X) \times (\Sigma \cup \{\varepsilon\}) \times 2^X \times L$ *is a finite set of edges, and* $\mathsf{Inv} : L \to I_M(X)$ *is the* invariant function*.*

We graphically represent timed automata in the usual way. The timed automaton represented in Figure 2.1 has one clock called x, three locations ℓ_0, ℓ_1 and ℓ_2. Location ℓ_0 is initial and location ℓ_1

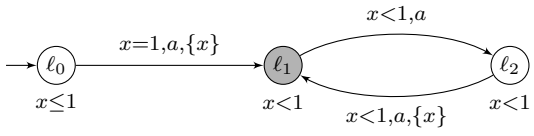


Figure 2.1: An example of timed automaton.

is final. The invariant of location ℓ_0 is $x \leq 1$. The alphabet of actions is the singleton $\{a\}$. The edge from location ℓ_2 to ℓ_1 is labeled with a guard $x < 1$, an action a and a reset of x. Sometimes in the document, there is no invariant (they are all true), hence they are omitted in the definitions and we define timed automata as tuples $(L, L_0, F, \Sigma, X, M, E)$.

Definition 2.2 (Semantics of timed automata)**.** *The semantics of a timed automaton* $\mathcal{A} = (L, L_0, F, \Sigma, X, M, E, \mathsf{Inv})$ *is given as a timed transition system* $\mathcal{T}_{\mathcal{A}} = (S, S_0, S_F, (\mathbb{R}_+ \cup (\Sigma \cup \{\varepsilon\})), \to)$ *where* $S = L \times \mathbb{R}_+^X$ *is the set of* states*,* $S_0 = L_0 \times \{\overline{0}\}$ *is the set of* initial states*,* $S_F = F \times \mathbb{R}_+^X$ *is the set of* final states*, and* $\to \subseteq S \times (\mathbb{R}_+ \cup (\Sigma \cup \{\varepsilon\})) \times S$ *is the* transition relation *composed of the following moves:*

- Discrete moves*:* $(\ell, v) \xrightarrow{a} (\ell', v')$ *for* $a \in \Sigma \cup \{\varepsilon\}$ *whenever there exists an edge* $(\ell, g, a, X', \ell') \in E$ *such that* $v \models g \wedge \mathsf{Inv}(\ell)$*,* $v' = v_{[X' \leftarrow 0]}$ *and* $v' \models \mathsf{Inv}(\ell')$*.*

- Time elapsing: $(\ell, v) \xrightarrow{\tau} (\ell, v + \tau)$ *for* $\tau \in \mathbb{R}_+$ *if* $v + \tau \models \mathsf{Inv}(\ell)$*.*

A timed automaton is said to be *complete* if for all actions $a \in \Sigma$ and for all states $(\ell, v) \in S$, there exists (ℓ', v') such that $(\ell, v) \xrightarrow{a} (\ell', v')$.

A *finite run* ϱ of $\mathcal{A}$ is a finite sequence of moves alternating time elapsing and discrete moves, starting in an initial state $s_0 \in S_0$ and ending with a discrete move labeled by an action in Σ, *i.e.*, $\varrho = s_0 \xrightarrow{\tau_0} s'_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{\tau_{k-1}} s'_{k-1} \xrightarrow{a_k} s_k$. We require that the last discrete move is not labeled by ε because we consider runs as readers of words. For example, $(\ell_0, 0) \xrightarrow{1} (\ell_0, 1) \xrightarrow{a} (\ell_1, 0) \xrightarrow{0.5} (\ell_1, 0.5) \xrightarrow{a} (\ell_2, 0.5) \xrightarrow{0.25} (\ell_2, 0.75) \xrightarrow{a} (\ell_1, 0)$ is a run of the timed automaton in Figure 2.1. For readability, we sometimes contract notations by writing $s \xrightarrow{\tau,a} s''$ instead of $s \xrightarrow{\tau} s' \xrightarrow{a} s''$.

The *reachability problem* asks, given a timed automaton $\mathcal{A}$ and a location ℓ of $\mathcal{A}$, whether there exists a finite run of $\mathcal{A}$ ending in ℓ.

An *infinite run* ϱ of $\mathcal{A}$ is an infinite sequence of moves alternating time elapsing and discrete moves, starting in an initial state $s_0 \in S_0$ and containing an unbounded number of discrete moves labeled by actions in Σ, *i.e.*, $\varrho = s_0 \xrightarrow{\tau_0} s'_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{\tau_{k-1}} s'_{k-1} \xrightarrow{a_k} s_k \cdots$. We require that infinite runs contain infinitely many discrete moves not labeled by ε because we consider infinite runs as readers of infinite timed words. For example, the sequence $(\ell_0, 0) \xrightarrow{1,a} (\ell_1, 0)\big(\xrightarrow{\frac{1}{3},a} (\ell_2, \frac{1}{3}) \xrightarrow{\frac{1}{3},a} (\ell_1, 0)\big)^\omega$ which endlessly alternates delays $\frac{1}{3}$ in ℓ_1 and ℓ_2, is an infinite run of the timed automaton in Figure 2.1.

Given an infinite run ϱ, we denote by $\varrho_{|n}$ the finite run defined as the *prefix* of the run ϱ ending with the n-th discrete transition different from ε.

Timed automata are often used as recognizers of languages. In the context of testing, one considers rather traces of the system whereas to study reachability, only runs as sequences of transitions are needed. The notion of languages is central in this document, hence we give definitions here.

Language of finite words of a timed automaton A finite run ϱ is said accepting if it ends in a state $s_k \in S_F$, that is in a final location. In the sequel, final locations are thus also said to be accepting and the other locations are said non-accepting. A *finite timed word* over Σ is a finite sequence $(t_i, b_i)_{i\leq n} \in (\mathbb{R}_+ \times \Sigma)^*$ such that $(t_i)_{i\leq n}$ is non-decreasing. We write $\mathcal{W}_\Sigma$, for the set of finite timed words over the alphabet Σ.

The timed word associated with a run $\varrho = s_0 \xrightarrow{\tau_0} s'_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{\tau_{k-1}} s'_{k-1} \xrightarrow{a_k} s_k$ is $w = (t_0, b_0) \dots (t_m, b_m)$ where $(b_j)_{j\leq m} \in \Sigma^{m+1}$ is the subsequence of the discrete moves of ϱ labeled in Σ and $t_j = \sum_{l=0}^{j} \tau_l$. In this case, w is said to be read by ϱ. The τ_i's are the delays elapsed between the actions a_i's along ϱ, whereas the t_i's are the absolute time where actions belonging to Σ (but not ε) are taken. We write $\mathcal{L}(\mathcal{A})$ for the *language* of $\mathcal{A}$, that is the set of timed words associated with an accepting run. For example, the language of the timed automaton in Figure 2.1 is the following set: $\big\{\big((1,a), \big((t_i,a)(t'_i,a)\big)_{0\leq i\leq n}\big) \in \mathcal{W}_{\{a\}} \mid (n \geq -1) \wedge (t'_0 < 2) \wedge (\forall 0 \leq i \leq n,\, t'_{i+1} - t'_i < 1)\big\}$. In words, this timed automaton accepts timed words starting by an action a at one time unit and then contains an even number of a's such that each pair of a's is done in less than one time unit.

Two timed automata $\mathcal{A}$ and $\mathcal{B}$ are said *equivalent* whenever $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. In order to compare behaviors of timed transition systems in a more precise way than the equality of languages, we introduce the notion of weak timed simulation relation.

Definition 2.3 (Weak timed simulation relation)**.** *A* weak timed simulation relation *between two timed transition systems $\mathcal{T}_i = (S^i, s^i_0, S^i_F, (\mathbb{R}_+ \cup (\Sigma \cup \{\varepsilon\})), \rightarrow_i)$ for $i \in \{1,2\}$ is a relation $\mathcal{R} \subseteq S^1 \times S^2$ such that:*

(1) $(s^1_0, s^2_0) \in \mathcal{R}$,

(2) for all $(s_1, s_2) \in \mathcal{R}$, *if* $s_1 \in S^1_F$ *then* $s_2 \in S^2_F$,

(3) for all $(s_1, s_2) \in \mathcal{R}$, *for all* $b \in \Sigma$ *whenever* $s_1 \xrightarrow{b}_1 s'_1$, *there exists* $s'_2 \in S^2$ *such that* $(s'_1, s'_2) \in \mathcal{R}$ *and* $s_2 \xrightarrow{b}_2 s'_2$,

(4) for all $(s_1, s_2) \in \mathcal{R}$, *whenever* $s_1 \xrightarrow{\tau^1_1}_1 \xrightarrow{\varepsilon}_1 s^1_1 \cdots \xrightarrow{\tau^1_{n-1}}_1 \xrightarrow{\varepsilon}_1 s^1_{n-1} \xrightarrow{\tau^1_n}_1 s'_1$, *there exists* $s'_2 \in S^2$ *such that* $(s'_1, s'_2) \in \mathcal{R}$ *and* $s_2 \xrightarrow{\tau^2_1}_2 \xrightarrow{\varepsilon}_2 s^2_1 \cdots \xrightarrow{\tau^2_{m-1}}_2 \xrightarrow{\varepsilon}_2 s^2_{m-1} \xrightarrow{\tau^2_m}_2 s'_2$ *with* $\sum_{i=1}^{n} \tau^1_i = \sum_{j=1}^{m} \tau^2_j$.

There exists alternative definitions of weak timed simulation relation, in the third item, sequences of the form $\varepsilon^*.b.\varepsilon^*$ (without delays between the actions) are considered instead of simply b. Our definition covers this case by applying the fourth item and considering only zero delays for the sequences of ε's.

If there is a weak timed simulation between $\mathcal{T}_{\mathcal{A}_1}$ and $\mathcal{T}_{\mathcal{A}_2}$, $\mathcal{A}_2$ is said to weak timed simulate $\mathcal{A}_1$. The intuition is that all the behaviors of $\mathcal{A}_1$ can be found in $\mathcal{A}_2$. The weak timed simulation of $\mathcal{A}_1$ by $\mathcal{A}_2$ implies, in particular, the language inclusion $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$. If the inverse relation of a weak timed simulation of $\mathcal{A}_1$ by $\mathcal{A}_2$ is also a weak timed simulation of $\mathcal{A}_2$ by $\mathcal{A}_1$, we talk about a weak timed bisimulation. In this case, languages are equal.

In a timed automaton, it is possible to find a timed word which is read by distinct runs. We then talk about non-determinism.

Definition 2.4 (Determinism)**.**

- *A deterministic timed automaton (abbreviated DTA) $\mathcal{A}$ is a TA such that for every finite timed word w, there is at most one run in $\mathcal{A}$ reading w.*

- *A timed automaton $\mathcal{A}$ is said* determinizable *if there exists a deterministic timed automaton $\mathcal{B}$ equivalent to $\mathcal{A}$.*

There is an alternative notion of deterministic timed automata which is more syntactic. It requires that from every location of the timed automaton and every action, guards of outgoing edges labeled with this action do not intersect between them. Our notion implies this one, it has the advantage to allow ε-transitions and the inconvenient is that it is harder to check whether a timed automaton is deterministic. Remark that the timed automaton in Figure 2.1 is clearly deterministic because in every location there is a single outgoing edge.

Language of infinite words of a timed automaton Similarly to the finite words case, one can define the acceptance of infinite words. The usual semantics to do so is the Büchi semantics. Then an infinite run is said accepting if it visits infinitely often the set of accepting locations F. An *infinite timed word* over Σ is an infinite sequence $(t_i, b_i)_{i\in\mathbb{N}} \in (\mathbb{R}_+ \times \Sigma)^{\mathbb{N}}$ such that $(t_i)_{i\in\mathbb{N}}$ is non-decreasing. We denote by $\mathcal{W}_\Sigma^\infty$ the set of infinite timed words over the alphabet Σ.

As previously, the timed word associated to a run $\varrho = s_0 \xrightarrow{\tau_0} s'_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{\tau_{k-1}} s'_{k-1} \xrightarrow{a_k} s_k \cdots$ is the subsequence of the discrete moves of ϱ labeled in Σ coupled with their absolute time. Thus, we write $\mathcal{L}^\infty(\mathcal{A})$ for the language of infinite timed words of $\mathcal{A}$. For instance, the language of infinite timed words of the timed automaton in Figure 2.1 is the set $\left\{\left((1,a), \left((t_i,a)(t'_i,a)\right)_{0\le i}\right) \in \mathcal{W}^\infty_{\{a\}} \mid (t'_0 < 2) \wedge (\forall 0 \le i,\, t'_{i+1} - t'_i < 1)\right\}$. In words, this timed automaton accepts infinite timed words over the alphabet $\{a\}$ starting by an a after one time unit and then, considering them two by two, each pair of a's is taken in less than one time unit.

Since runs are infinite, some convergence phenomena may appear. The most famous is the zenoness which simply corresponds to the convergence of the absolute time along a run.

Definition 2.5 (Zenoness)**.**

- *An infinite run $\varrho = s_0 \xrightarrow{\tau_0} s'_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{\tau_{k-1}} s'_{k-1} \xrightarrow{a_k} s_k \cdots$ is* Zeno *(resp.* non-Zeno*) if the sum $\sum_{i\in\mathbb{N}} \tau_i$ is bounded (resp. infinite).*
- *An infinite timed word $(t_i, b_i)_{i\in\mathbb{N}}$ is* Zeno *(resp.* non-Zeno*) if $(t_i)_{i\in\mathbb{N}}$ is finite (resp. diverges).*
- *A timed automaton is* strongly non-Zeno *if in every cycle $\ell_1 \to \ell_2 \cdots \to \ell_1$, there is one clock which is reset and lower guarded by a positive constant [AMPS98].*

Note that an infinite run is Zeno (resp. non-Zeno) if and only if its associated timed word is. Moreover, the strong non-zenoness is equivalent to the absence of Zeno run.

Region abstraction Given the maximal constant M of a timed automaton $\mathcal{A} = (S, S_0, S_F, (\mathbb{R}_+ \cup (\Sigma \cup \{\varepsilon\})), \to)$, the usual region abstraction introduced in [AD94] forms a partition of the valuations over X. In the following definition, $\lfloor t \rfloor$ and $\{t\}$ are respectively the integer part and the fractional part of the real t.

Definition 2.6 (Region equivalence)**.** *The* region equivalence $\equiv_\mathcal{A}$ *over valuations of X is defined as follows: $v \equiv_\mathcal{A} v'$ if*

1. *for every clock $x \in X$, $v(x) \le M$ iff $v'(x) \le M$;*
2. *for every clock $x \in X$, if $v(x) \le M$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ and $\{v(x)\} = 0$ if and only if $\{v'(x)\} = 0$*

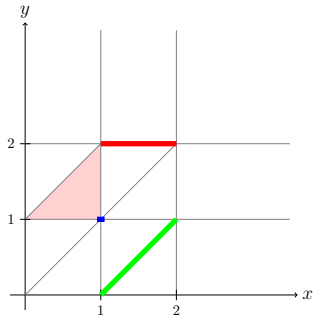


Figure 2.2: Illustration of the region construction.

3. *for every pair of clocks* $(x, y) \in X^2$ *such that* $v(x) \leq M$ *and* $v(y) \leq M$, $\{v(x)\} \leq \{v(y)\}$ *if and only if* $\{v'(x)\} \leq \{v'(y)\}$.

The finitely many equivalence classes of this relation are called *regions* and Reg_M^X denotes the set of regions for the timed automaton $\mathcal{A}$. For each valuation v of the clocks of $\mathcal{A}$, there is a single region containing v, denoted by $R(v)$. A region R' is a *time-successor* of a region R if there exists $v \in R$ and $t \in \mathbb{R}_+$ such that $v + t \in R'$ and $R' \neq R$. For example, for two clocks x and y and the maximal constant 2, the regions form the partition represented on Figure 2.2. Regions are of several forms, for example, bounded regions can be triangles, segments (horizontal, vertical or diagonal), or points. Note that the colored horizontal segment in Figure 2.2 is a time-successor of the colored triangle. In the sequel, we write $\overrightarrow{R}$ for the set of time-successors of a region R.

Remark that, by definition of the region equivalence, if a region over a set of clocks X is projected over a subset of clocks $X' \subseteq X$, the resulting set of valuations is a region over X'. In particular, in any region, the set of values of a given clock is a singleton $\{k\}$ with $k \in \mathbb{N}$, or an interval of the form $(k, k+1)$ with $k \in \mathbb{N}$, or $(M, +\infty)$.

Thanks to the region equivalence, from any timed automaton, one can build a finite automaton called the region automaton which allows to decide properties such as reachability for this timed automaton.

Definition 2.7 (Region automaton)**.** *Let* $\mathcal{A} = (L, L_0, F, \Sigma, X, M, E, \mathsf{Inv})$ *be a timed automaton. The* region automaton $\mathsf{RG}(\mathcal{A})$ *of* $\mathcal{A}$ *is the transition system* $(\mathsf{S}, \mathsf{S}_0, \mathsf{S}_F, \Sigma, \to)$ *where states in* S *are pairs* (ℓ, r) *with* $\ell \in L$ *and* $r \in \mathsf{Reg}_M^X$*, initial states* $\mathsf{S}_0 = L_0 \times \{\overline{0}_X\}$ *are made of an initial location and the region of the initial valuation, final states* S_F *are pairs* (ℓ, r) *such that* $\ell \in F$ *and* $r \in \mathsf{Reg}_M^X$*, and for all* $\ell, \ell' \in L$*, for all* $r, r' \in \mathsf{Reg}_M^X$ *and for all* $a \in \Sigma \cup \{\varepsilon\}$ *there is a transition* $(\ell, r) \xrightarrow{a} (\ell', r')$ *if there exist two valuations* $v \in r$*,* $v' \in r'$ *and a delay* $d \in \mathbb{R}_+$ *such that* $(\ell, v) \xrightarrow{d} (\ell, v + d) \xrightarrow{a} (\ell', v')$ *is a sequence of moves in* $\mathcal{A}$*.*

Figure 2.3 illustrates the construction of the region automaton for the timed automaton in Figure 2.1. Behaviors of region automata are tightly linked to behaviors of timed automata which are abstracted. In particular, a run in a timed automaton can be projected in the region automaton as follows. Given a run $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \cdots (\ell_{k-1}, v_{k-1}) \xrightarrow{\tau_{k-1}, a_{k-1}} (\ell_k, v_k) \cdots$, the *projection* π_ϱ of ϱ is $\pi_\varrho = (\ell_0, R(v_0)) \xrightarrow{\tau_0, a_0} (\ell_1, R(v_1)) \cdots (\ell_{k-1}, R(v_{k-1})) \xrightarrow{\tau_{k-1}, a_{k-1}} (\ell_k, R(v_k)) \cdots$. For example, the projection of $(\ell_0, 0) \xrightarrow{1,a} (\ell_1, 0) \xrightarrow{0.5,a} (\ell_2, 0.5) \xrightarrow{0.25,a} (\ell_1, 0)$ is $(\ell_0, \{0\}) \xrightarrow{a} (\ell_1, \{0\}) \xrightarrow{a} (\ell_2, (0, 1)) \xrightarrow{a} (\ell_1, \{0\})$.

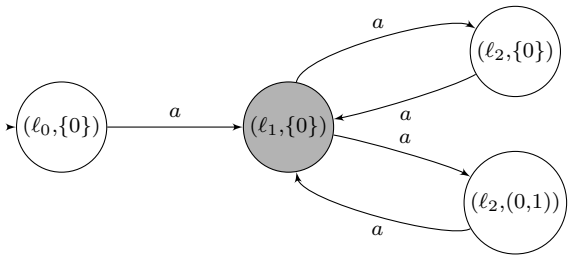


Figure 2.3: The region automaton of the timed automaton in Figure 2.1.

More generally, the finite automaton $\mathsf{RG}(\mathcal{A})$ time-abstract simulates the timed automaton $\mathcal{A}$, that is $\mathsf{RG}(\mathcal{A})$, seen as a timed automaton with no clock, weakly timed simulates $\mathcal{A}$.

Diagonal constraints In the sequel, we sometimes abuse notations, considering regions as a conjunction of constraints instead of sets of valuations. Most regions cannot be expressed as the set of valuations satisfying a conjunction of constraints of the form $x \sim c$ with $x \in X,\ c \in [0, M] \cap \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$, because the order over the fractional parts of the clocks impacts. As a consequence, we also need constraints of the form $x - y \sim c$ where $x, y \in X,\ c \in [0, M] \cap \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. However, the model of timed automata can be enriched with guards containing diagonal constraints. The resulting model has the same expressiveness as the original one. Indeed, diagonal constraints can be removed by building as much copies of the timed automaton as sets of diagonal constraints (see for example [Bou09]). The set of diagonal constraints which are satisfied in a state is thus encoded by the copy to which the location belongs. Edges go from one copy to the other, depending on the resets which can change the set of satisfied diagonal constraints. The construction is correct because time elapsing preserves the set of diagonal constraints which are satisfied. Unfortunately, this construction may introduce an exponential blow-up in the size of the model.

A timed version of the region automaton The region automaton can be seen in a finer way as a timed automaton by labeling transitions with the corresponding guards and the corresponding resets and restoring the invariants associated with locations. A transition of the form $(\ell, r) \xrightarrow{a} (\ell', r')$ thus becomes $((\ell, r), \mathsf{Inv}(\ell)) \xrightarrow{r'', a, X'} ((\ell', r'), \mathsf{Inv}(\ell'))$ where r'' is the time-successor of r equal, after resets of clocks by the transition, to r'. As explained above, diagonal constraints introduced by the region guards could be removed inducing a potential exponential blow-up in the size of the automaton. However, the satisfaction of the diagonal constraints of the region guards is already encoded in regions of the locations. As a consequence, diagonal constraints are useless and can be removed without the usual construction.

The resulting timed automata weak timed bisimulates $\mathcal{A}$. This is easily proved considering the relation $\equiv \subseteq (L \times \mathbb{R}^X) \times \big((L \times Reg_{\mathcal{A}}) \times \mathbb{R}^X\big)$ defined as follows: $\Big((\ell, v), \big((\ell', r'), v'\big)\Big) \in \equiv$ if $v = v'$, $\ell = \ell'$ and there exists $r \in \overrightarrow{r'}$ such that $v \in r$.

Part III

Determinization of Timed Automata

Introduction

A timed automaton is said non-deterministic if it has at least two runs reading the same timed word. The determinization, that is, the construction of an equivalent deterministic timed automaton, is used to address several problems such as implementability, diagnosis or test generation, where the underlying analyses depend on the observable behavior. For example, in the context of testing, the specification has to be determinized in some sense. Indeed, we need to foresee the allowed outputs after a trace (*i.e.* a sequence of observations), thus the set of states after this trace. More generally, restriction to deterministic timed automata makes a lot of problems simpler. In particular, a deterministic timed automaton can easily be completed and then complemented by exchanging accepting and non-accepting locations. This is for instance useful for model-checking. Indeed, given a deterministic timed automaton $\mathcal{A}_\varphi$ representing a property, one can easily decide whether an other timed automaton satisfies the property by performing the intersection with the complement of $\mathcal{A}_\varphi$, and then checking the emptiness of the language of the resulting timed automaton. This approach has been used in [BDL$^+$12] for the model-checking of weighted metric temporal logic, that is a logic that can express properties such as "Can we reach a target location in less than 10 time units and with a cost less than 4?".

For finite automata, determinization can be performed by a subset construction. The successor of a state by an action naturally becomes the set of its successors by this action. For timed automata it is not as simple because of resets. If two runs read the same word, one where a clock is reset along the first transition and another where it is not, then for the determinization, we need to preserve the clock information, hence we intuitively need an additional clock. For example, the timed automaton in Figure 2.4 is not determinizable. In fact, it is not complementable which implies non-determinizability.

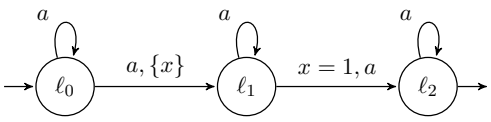


Figure 2.4: A non-determinizable timed automaton from [AD94].

This timed automaton accepts runs reading two a's separated exactly by one time unit. An unbounded number of a's may happen in a single time unit. Thus, the complement would need an unbounded number of clocks to check whether an a does not occur exactly one time unit after each a. As a consequence, determinizable timed automata form a strict subclass of timed automata [AD94]. Moreover, the determinizability of timed automata is undecidable [Tri06], even with fixed resources [Fin06]. That is, given a fixed number of clocks, a maximal constant and a timed automaton, one cannot decide whether there exists an equivalent deterministic timed automaton with this number of clocks and this maximal constant.

Nevertheless, some timed automata classes are known to be determinizable. A first example of determinizable class is the one of event-clock timed automata [AFH94], that is timed automata such that each clock is associated with an action and is reset exactly when this action occurs. More generally, timed automata whose resets do not depend on the run but only on the input word are determinizable by performing the classical subset construction. Another example is the class of timed automata with integer resets, that is timed automata in which resets occur only on edges whose guards are punctual (*i.e.* contain a constraint of the form $x = c$) [SPKM08]. Finally, strongly non-Zeno timed automata, that is timed automata in which accumulated delays diverges along all runs, are also determinizable [AMPS98].

To overcome the non-feasibility of determinization of timed automata in general, two alternatives have been explored: either exhibiting subclasses of timed automata which are determinizable and provide ad hoc determinization algorithms, or constructing deterministic over-approximations. We relate below, for each of these directions, a recent contribution.

- **A pseudo-algorithm** An abstract determinization procedure which effectively constructs a deterministic timed automaton for several classes of determinizable timed automata is presented in [BBBB09]. Given a timed automaton $\mathcal{A}$, this procedure first produces a language-equivalent infinite timed tree, by unfolding $\mathcal{A}$, introducing a fresh clock at each step. This allows one to preserve all timing constraints, using a mapping from clocks of $\mathcal{A}$ to the new clocks. Then, the infinite tree is split into regions, and symbolically determinized. Assuming that at each level of the tree, only a finite number of clocks is used (this condition is the clock-boundedness assumption), the infinite tree with infinitely many clocks can be folded up into a timed automaton (with finitely many locations and clocks). The clock-boundedness assumption is satisfied by the syntactic determinizable classes of timed automata mentioned above, which can thus be determinized by this procedure. The resulting deterministic timed automaton is doubly exponential in the size of $\mathcal{A}$: more precisely, the number of locations of the resulting automaton is doubly exponential in its number of clocks and in the number of clocks of $\mathcal{A}$ and exponential in the number of locations of $\mathcal{A}$.

- **An over-approximation** By contrast, Krichen and Tripakis propose in [KT09] an algorithm applicable to any timed automaton $\mathcal{A}$, which produces a deterministic over-approximation, that is a deterministic timed automaton $\mathcal{B}$ accepting at least all timed words in $\mathcal{L}(\mathcal{A})$. Given a set of new clocks, the timed automaton $\mathcal{B}$ is built by simulation of $\mathcal{A}$ using only information stored in the new clocks. A location of $\mathcal{B}$ constitutes a state estimate of $\mathcal{A}$ consisting of a (generally infinite but finitely represented) set of pairs (ℓ, v) where ℓ is a location of $\mathcal{A}$ and v a valuation over the union of clocks of $\mathcal{A}$ and $\mathcal{B}$. This method is based on the use of a fixed finite automaton (called a skeleton) which governs the resetting policy for the clocks of $\mathcal{B}$. This policy is *a priori* fixed, a deterministic timed automaton could thus be strictly over-approximated by this approach. Here also the timed automaton $\mathcal{B}$ is doubly exponential in the size of $\mathcal{A}$.

Contribution: a new algorithm In this part, we propose to improve both approaches in a single one. To do so, we combine advantages of each of them. We want a procedure which exactly determinizes at least the known determinizable classes, but which always terminates yielding a deterministic timed automaton (potentially approximate). The heart of the problem is to find a good reset policy. Hence, we propose to build a game using the state estimate of [KT09], in which a player called "Determinizator" chooses the resets while the opponent "Spoiler" choses the action and its timing. A winning strategy for Determinizator then corresponds to an exact determinization. A losing strategy yields a deterministic over-approximation.

- **Inspired by a game approach for fault diagnosis** This approach is inspired by a game approach which has been developed for fault diagnosis by deterministic timed automata [BCD05]. The problem considered is the synthesis of fault diagnosers which are realizable with deterministic timed automata over fixed resources (*i.e.* number of clocks and granularity of clocks). Given some resources, a game is built with the following property: there is a winning strategy for the game if and only if there exists a deterministic diagnoser over these resources. Moreover, the winning strategy can be turned into such a diagnoser.

- **Extension of our approach to deal with ε-transitions and invariants** We define extensions of the approach to deal with ε-transitions and invariants preserving the properties of the results. These extensions are particularly interesting for the application to test selection. Indeed, they respectively allow to model partial observability and urgency which are primordial features for specifications.

- **Beyond the over-approximation** Depending on the context, the over-approximation of languages is not the suitable approximation. We thus propose another extension of the approach which combines under- and over-approximations. We consider timed automata with an alphabet partitioned in two sub-alphabets. Roughly, transitions over actions in the first one are under-approximated, and transitions over actions of the second one are over-approximated. Once again, this extension is, in particular, motivated by the application to testing where inputs and outputs of the systems need to be treated in different ways.

- **Application to model-based conformance testing** We then focus on the application of our approach for the determinization of timed automata to model-based conformance testing.

  - **Model-based conformance testing** Roughly, the problem asks, given the model of a specification and a black-box implementation, assumed to behave like an unknown timed automaton, whether this implementation conforms to the specification for a given conformance relation. To do so, a tester interacts with the black-box sending inputs, observing the outputs and exploring the specification to check whether these outputs are allowed, and finally emits a verdict: pass or fail. Testing does not permit to prove the conformance of an implementation, it is rather used to detect non-conformances. As a consequence, the primordial property of test cases is soundness, *i.e.* a fail verdict occurs only in case of non-conformance.
  - **Our setting** In our context, specifications are given as non-deterministic timed automata and the conformance relation is the classical **tioco** relation. Roughly, an implementation conforms to a specification if, after a common trace, outputs of the implementation belong to the set of enable outputs of the specification. The need for determinization thus appears when we want to emit the right verdict (*i.e.* fail verdict should imply to non-conformance). Indeed, to compute the enable outputs of the specification after a fixed trace, either the specification is determinized *a priori* or it is determinized on-the-fly along a trace. The drawback of the on-the-fly computation is that it could be incompatible with real-time testing, taking too much time for decisions.
  - **Our contribution** We propose a general framework for the off-line generation of test cases from specifications given as non-deterministic timed automata. The determinization of timed automata being impossible in general, we tackle the determinization problem using our game approach with its extensions. Inputs are under-approximated to specify less, and outputs are over-approximated to allow more behaviors. Let us consider S' the

resulting deterministic timed automaton from our approach applied to a non-deterministic specification S where both alphabets are respectively inputs and outputs. Then, the main property of our approach is that test cases soundly generated from S' are also sound for S, even if the determinization was approximated. This is due to the fact that our combination of under- and over-approximation preserves the conformance relation.

Comparisons with existing work can be found in Chapter 4. In particular, we discuss the differences with the framework of [KT09] which also uses an approximate determinization. The main point is that the urgency of the specification is totally removed by the determinization in [KT09] but not by ours.

Outline This part is structured in two chapters. Our game approach for the determinization is defined with its extensions and compared to existing methods in Chapter 3. Then, Chapter 4 is devoted to the application of this approach to off-line test selection.

Chapter 3

A Game Approach to Determinize Timed Automata

Introduction

Determinization of timed automata is not possible in general [AD94], moreover the problem of the determinizability of a timed automaton is undecidable [Fin06, Tri06]. In the introduction, we presented some classes of determinizable timed automata and two existing approaches for the determinization. First, a procedure which determinizes all the timed automata belonging to known determinizable classes, but which does not terminate in general [BBBB09]. Second, an algorithm which, given a timed automaton and fixed resources, yields a deterministic over-approximation over these resources [KT09].

A very important aspect of the latter approach is the use of an *a priori* fixed policy for the resets. Indeed, finding a good policy for resets is the key for determinization. Thanks to this observation, we propose a method that improves the approaches of [BBBB09] and [KT09], despite their notable differences. It is inspired by a game-based approach to decide the diagnosability of timed automata with fixed resources presented by Bouyer, Chevalier and D'Souza in [BCD05]. Similarly to [KT09], in our approach, the resulting deterministic timed automaton is given fixed resources (number of clocks and maximal constant) in order to simulate the original timed automaton by a coding of relations between new clocks and original ones. The core principle is the construction of a finite turn-based safety game between two players, Spoiler and Determinizator, where Spoiler chooses an action and the region of its occurrence, while Determinizator chooses which clocks to reset. Our main result states that if Determinizator has a winning strategy, then it yields a deterministic timed automaton accepting exactly the same timed language as the initial automaton, otherwise it produces a deterministic over-approximation.

Our approach is more general than the procedure of [BBBB09], thus allowing one to enlarge the set of timed automata that can be automatically determinized, thanks to an increased expressive power in the coding of relations between new and original clocks, and robustness to some language inclusions (*e.g.* a non-determinizable sub-automaton can be ignored if its language is included in the one for the rest of the timed automaton). Moreover, in contrast to [BBBB09], our technique applies to a larger class of timed automata: timed automata with ε-transitions and invariants. It is also more precise than the algorithm of [KT09] in several respects: an adaptative and timed resetting policy, governed by a strategy, compared to a fixed untimed one, and a more precise update of the relations between clocks, even for a fixed policy, allow our method to remain exact on a larger class of timed

automata. The model used in [KT09] includes silent transitions, and edges are labeled with urgency status (eager, delayable, or lazy), but urgency is not preserved by their over-approximation algorithm. These observations illustrate the benefits of our game-based approach compared to existing work.

Another contribution is the generalization of our game-based approach to generate deterministic under-approximations or, more generally, deterministic approximations combining under- and over-approximations. The motivation for this generalization is to tackle the problem of off-line model-based test generation from non-deterministic timed automata specifications [BJSK12], presented in Chapter 4. Indeed in this context, input actions and output actions have to be considered differently while building the approximation. We provide a notion of refinement (and the dual abstraction) and explain how to extend our approach to generate deterministic abstractions.

This chapter is based on the eponymous paper [BSJK11a] and its research report [BSJK11b]. Its structure is as follows: Section 3.1 is devoted to the presentation of our game approach and its properties. A comparison with existing methods is detailed in Section 3.2. Extensions of the method to timed automata with invariants and ε-transitions are then presented in Section 3.3. In Section 3.4, we finally discuss how the construction can be adapted to perform under-approximations, or combinations of under- and over-approximations.

3.1 The game approach

Intuitively, the subset construction, which successfully determinize finite automata, fails for timed automata because of non uniform resets. When performing a subset construction, it could thus be necessary to use an unbounded number of clocks to store information from all possible paths so far. This phenomenon on a particular timed automaton indicates that it is not determinizable by the approach of [BBBB09]. Our objective is to design a finer approach, yet the main problem remains to find sufficient resources (*i.e.* a number of clocks and a maximal constant for the guards) and suitable resets to preserve all the timing information needed for the subset construction. One key feature of our approach lies in the use of relations between the clocks (*i.e.* unions of regions corresponding to conjunctions of atomic diagonal constraints) to encode the important timing information.

In [BCD05], given a plant —modeled by a timed automaton— and fixed resources, the authors build a game where one player has a winning strategy if and only if the plant can be diagnosed by a timed automaton using the given resources. Inspired by this construction, given a timed automaton $\mathcal{A}$, over some resources, and given fixed resources (k, M'), we derive a game between two players Spoiler and Determinizator, such that if Determinizator has a winning strategy, then a deterministic timed automaton $\mathcal{B}$, over resources (k, M'), with $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})$ can be effectively generated. Moreover, any strategy for Determinizator (winning or not) yields a deterministic over-approximation for $\mathcal{A}$. For clarity, we first expose the method for timed automata without ε-transitions and in which all invariants are true. The general case is presented as an extension, in Section 3.3.

3.1.1 Game definition

Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ be a timed automaton. The resources of $\mathcal{A}$ are $(|X|, M)$. We aim at building a deterministic timed automaton $\mathcal{B}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ if possible, or $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. In order to do so, we fix resources (k, M') for $\mathcal{B}$ and build a finite 2-player turn-based safety game $\mathcal{G}_{\mathcal{A},(k,M')}$. Players Spoiler and Determinizator alternate moves, and the objective of player Determinizator is to avoid a set of bad states (to be defined later). Intuitively, in the safe states, for sure, no over-approximation has been performed.

One consider timed automata with a single initial location to simplify notations. It is not a restriction because we have ε-transitions. For simplicity, we first detail the approach in the case where $\mathcal{A}$ has no ε-transitions and all invariants are true. Note that the definition can be difficult to read, but some details of the construction of the game for the small timed automaton in Figure 3.1 with a single clock are then given to illustrate the different steps.

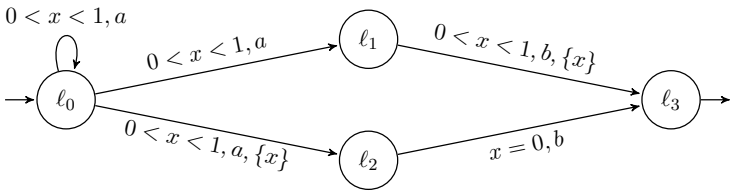


Figure 3.1: A timed automaton $\mathcal{A}$.

Let Y be a set of clocks, disjoint from X, and of cardinality k. This is the set of clocks of $\mathcal{B}$ which encodes X, the set of clocks of $\mathcal{A}$, through relations over $X \cup Y$. Formally, given X a set of clocks, a *relation* C over X is the union of the regions intersecting a given conjunction of atomic constraints of the form $x - y \sim c$, where $x, y \in X$, $\sim \in \{<, =, >\}$ and $c \in \mathbb{N}$. When all constants c belong to $[-M, M]$ for some constant $M \in \mathbb{N}$ we denote by $\mathsf{Rel}_M(X)$ for the set of relations over X.

The idea is to perform a subtle subset construction using relations to try to determinize $\mathcal{A}$. Using union of regions instead of simple conjunctions of diagonal constraints allows to restrict their expressiveness with a maximal constant to have a finite number of possible relations, in the same way as for regions. In the examples of the sequel, we often abuse notations writing conjunctions of constraints, for readability, instead of unions of regions. For instance, $\forall z, z' \in X, z - z' = 0$ represents the union of the regions of the forms $\{k\}^X$ and $(k-1, k)^X$ with k an integer smaller than M, but also with the region $(M, \infty)^X$. Relations are updates in the construction of the game using the operation $\overleftrightarrow{R}$ which assigns to a union of regions R, the union of the time-successors and time-predecessors of regions in R.

States of the game (future locations of the resulting deterministic timed automaton) are state estimates, symbolically represented using locations of $\mathcal{A}$ and regions over clocks in Y together with relations for the clocks in $X \cup Y$. The risk of over-approximation is marked and propagated thanks to booleans.

The initial state of the game is a state of Spoiler consisting of a single configuration with location ℓ_0 (initial location of $\mathcal{A}$), the simplest relation over $X \cup Y$: $\forall z, z' \in X \cup Y, z - z' = 0$, and the marking $\top$ (no over-approximation was done so far), together with the null region over Y.

In each of its states, Spoiler challenges Determinizator by proposing an M'-bounded region r over Y, and an action $a \in \Sigma$, representing that Spoiler chooses to read an a in the region r. Determinizator answers by deciding the set of clocks $Y' \subseteq Y$ it wishes to reset. The next state of Spoiler contains a region over Y ($r' = r_{[Y' \leftarrow 0]}$), and a finite set of configurations: triples formed of a location of $\mathcal{A}$, a relation on clocks in $X \cup Y$, and a boolean marking ($\top$ or $\bot$). A state of Spoiler thus constitutes a state estimate of $\mathcal{A}$, and the role of the markings is to indicate whether over-approximations possibly happened. A state of Determinizator is a copy of the preceding state estimate of Spoiler together with the move of Spoiler.

Bad states player Determinizator wants to avoid are, on the one hand states of the game where all configurations are marked $\bot$ and, on the other hand, states where all final configurations (if any) are marked $\bot$.

Definition 3.1. *Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ be a timed automaton and (k, M') resources. We let $\mathbf{M} = \max(M, M')$, and Y a set of k clocks. The game associated with $\mathcal{A}$ and (k, M') is $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ where:*

- V *is a finite set of vertices, partitioned into V_S (vertices of Spoiler) and V_D (vertices of Determinizator). $\mathsf{V}_S \subseteq 2^{L \times \mathsf{Rel}_{\mathbf{M}}(X \cup Y) \times \{\top, \bot\}} \times \mathsf{Reg}^Y_{M'}$ and $\mathsf{V}_D \subseteq \mathsf{V}_S \times \mathsf{Reg}^Y_{M'} \times \Sigma$;*
- $\mathsf{v}_0 = (\{(\ell_0, \bigwedge_{z,z' \in X \cup Y} z - z' = 0, \top)\}, \{\overline{0}\})$ *is the initial vertex; $\mathsf{v}_0 \in \mathsf{V}_S$;*
- Act *is the set of possible actions partitioned into $\mathsf{Act}_S = \mathsf{Reg}^Y_{M'} \times \Sigma$ and $\mathsf{Act}_D = 2^Y$;*
- $\delta = \delta_S \cup \delta_D$ *is the transition relation with δ_S and δ_D defined as follows*
  - $\delta_S \subseteq \mathsf{V}_S \times \mathsf{Act}_S \times \mathsf{V}_D$ *consists of all edges of the form $(\mathcal{E}, r) \xrightarrow{(r',a)} ((\mathcal{E}, r), (r', a))$ if $r' \in \overrightarrow{r}$ and there exist $(\ell, C, b) \in \mathcal{E}$ and $\ell \xrightarrow{g,a,X'} \ell' \in E$ such that*

$$[r' \cap C]_{|X} \cap g \neq \emptyset \ ; \tag{$\mathsf{C}_{\neq\emptyset}$}$$

  - $\delta_D \subseteq \mathsf{V}_D \times \mathsf{Act}_D \times \mathsf{V}_S$ *is the set of edges of the form $((\mathcal{E}, r), (r', a)) \xrightarrow{Y'} (\mathcal{E}', r'_{[Y' \leftarrow 0]})$ where $\mathcal{E}' = \cup_{\gamma \in \mathcal{E}} \mathsf{Succ}_e[r', a, Y'](\gamma)$ where:*
    - Succ_e *is the elementary successor function*

$$\mathsf{Succ}_e[r', a, Y'](\ell, C, b) = \left\{(\ell', C', b') \mid \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ s.t. } \begin{cases} [r' \cap C]_{|X} \cap g \neq \emptyset \\ C' = \mathsf{up}(r', C, g, X', Y') \\ b' = b \wedge ([r' \cap C]_{|X} \subseteq g) \end{cases}\right\}, \tag{3.1}$$

with up *the update function for relations*

$$\mathsf{up}(r', C, g, X', Y') = \overleftarrow{(r' \cap C \cap g)_{[X' \leftarrow 0][Y' \leftarrow 0]}} \ ; \tag{3.2}$$

- $\mathsf{Bad} \subseteq \mathsf{V}_S$ *is the set of bad vertices, defined by*
  $\mathsf{Bad} = \big\{(\{(\ell_j, C_j, \bot)\}_j, r)\big\} \cup \big\{(\{(\ell_j, C_j, b_j)\}_j, r) \,|\, \{\ell_j\}_j \cap F \neq \emptyset \wedge \forall j,\, \ell_j \in F \Rightarrow b_j = \bot\big\}.$

The objective of the game for player Determinizator is to avoid the set Bad*. Player Spoiler has the opposite objective.*

Let us comment the definition of the game. The edge relation δ gives the possible moves for each player and is deterministic: for every $\mathsf{v}_S \in \mathsf{V}_S$ and $(r', a) \in \mathsf{Act}_S$ there is a single successor vertex $\mathsf{v}_D \in \mathsf{V}_D$ such that $(\mathsf{v}_S, r', a, \mathsf{v}_D) \in \delta$ and for every $\mathsf{v}_D \in \mathsf{V}_D$ and $Y' \in \mathsf{Act}_D$ there is a single successor vertex $\mathsf{v}_S \in \mathsf{V}_S$ such that $(\mathsf{v}_D, Y', \mathsf{v}_S) \in \delta$. We now detail how these successors are defined.

A state $\mathsf{v}_S = (\mathcal{E}, r) \in \mathsf{V}_S$ is composed of a state estimate $\mathcal{E}$ together with a region over Y, and elements of $\mathcal{E}$ are called *configurations*. Given $\mathsf{v}_S = (\mathcal{E}, r) \in \mathsf{V}_S$ a state of Spoiler and $(r', a) \in \mathsf{Act}_S$ one of its moves, the successor state is defined, provided r' is a time-successor of r, as the state $\mathsf{v}_D = (\mathcal{E}, (r', a)) \in \mathsf{V}_D$ if the successors of this state have a non-empty set of configurations.

Given $\mathsf{v}_D = (\mathcal{E}, (r', a)) \in \mathsf{V}_D$ a state of Determinizator and $Y' \in \mathsf{Act}_D$ one of its moves, the successor state of v_D is the state $(\mathcal{E}', r'_{[Y' \leftarrow 0]}) \in \mathsf{V}_S$ where $\mathcal{E}'$ is obtained as the set of all elementary successors of configurations $(\ell, C, b) \in \mathcal{E}$ by move (r', a) and after resetting Y': $\mathcal{E}' =$

$\{\mathsf{Succ}_e[r', a, Y'](\ell, C, b) \mid (\ell, C, b) \in \mathcal{E}\}$. The formal definition of the elementary successor function is given above in Equation (3.1) for a configuration (ℓ, C, b).

$\mathsf{up}(r', C, g, X', Y')$ is the update of the relation on clocks in $X \cup Y$ after the moves of the two players, that is after taking action a in r', resetting $X' \subseteq X$ and $Y' \subseteq Y$, and ensuring the satisfaction of g. The resulting updated relation is also formally defined above, in Equation (3.2), as a union of regions on $X \cup Y$ with maximal constant $\mathbf{M}$. In the update, the intersection with g aims at stopping runs that for sure will correspond to timed words outside of $\mathcal{L}(\mathcal{A})$. Region r', relation C and guard g can all be seen as zones (*i.e.* unions of regions) over clocks $X \cup Y$. It is standard that elementary operations on union of regions, such as intersections, resets, future and past, can be performed effectively. As a consequence, the update of a relation can also be computed effectively.

The boolean b keeps track of potential over-approximations. Boolean b' is set to $\bot$ if either $b = \bot$ or the induced guard $[r' \cap C]_{|X}$ over-approximates g: $\neg([r' \cap C]_{|X} \subseteq g)$ (this condition is written $\mathsf{C}_{\neg\subseteq}$ for short).

Size of $\mathcal{G}_{\mathcal{A},(k,M')}$. State estimate are sets of configurations, which contain each a relation over $X \cup Y$. Therefore, the number of states in $\mathcal{G}_{\mathcal{A},(k,M')}$ is doubly exponential in the size of $\mathcal{A}$. Also, Determinizator states have exponentially many outgoing edges in k, the size of Y. We will see in Proposition 3.1 that the number of edges in $\mathcal{G}_{\mathcal{A},(k,M')}$ can be impressively decreased, since restricting to atomic resets (resets of at most one clock at a time) does not diminish the power of Determinizator. Nevertheless, the complexity order is not impacted and the size of the resulting deterministic timed automaton could even be larger than with multiple resets.

3.1.2 Example

As an example, the construction of the game is illustrated on the non-deterministic timed automaton $\mathcal{A}$ depicted in Figure 3.1, page 43. Part of the construction of the associated game $\mathcal{G}_{\mathcal{A},(1,1)}$ is represented in Figure 3.2. $\mathcal{A}$ has a single clock called x, and the game uses a single clock y (for simplicity, but the construction would work with an arbitrary number of clocks). Rectangular states belong to Spoiler and circular ones to Determinizator. Note that, to simplify the picture, the labels of states of Determinizator are omitted (recall that they contain the predecessor state together with the move of Spoiler).

Let us detail the first steps of the game construction. The initial state v_0 contains the single configuration $(\ell_0, x - y = 0, \top)$ together with the initial region $\{0\}$ over $\{y\}$. To determine the possible moves of Spoiler from the initial state, we examine the outgoing transitions from ℓ_0 in $\mathcal{A}$: they all read letter a and are guarded by $0 < x < 1$. Given the relation $x - y = 0$, this guard can be expressed using clock y without any approximation: $0 < y < 1$. Thus, the only possible move for Spoiler from the initial state of $\mathcal{G}_{\mathcal{A},(1,1)}$ is $(0 < y < 1, a)$ and the successor configurations will still have $\top$ as boolean, reflecting that no over-approximation happened so far (condition $\mathsf{C}_{\neg\subseteq}$ is not fulfilled). The successor state by this move is a state of Determinizator, defined as the pair formed of its predecessor state v_0 and the move of Spoiler $((0, 1), a)$.

From there, Determinizator has two possible moves: resetting y or not. We explain the computation of the successor configurations by the move $\emptyset$ of Determinizator, yielding the state v_1. Since in $\mathcal{A}$, from location ℓ_0, there are three transitions corresponding to the move $(0 < y < 1, a)$ of Spoiler, three successor configurations need to be computed. The transitions to locations ℓ_0 and ℓ_1 do not reset x, and Determinizator chose not to reset y, thus the relation for the corresponding configurations still is $x - y = 0$. For the last configuration, associated with the target location ℓ_2, x is reset in $\mathcal{A}$, but y isn't and $0 < y < 1$, so we derive the relation $-1 < x - y < 0$. Recall that the markers of all the

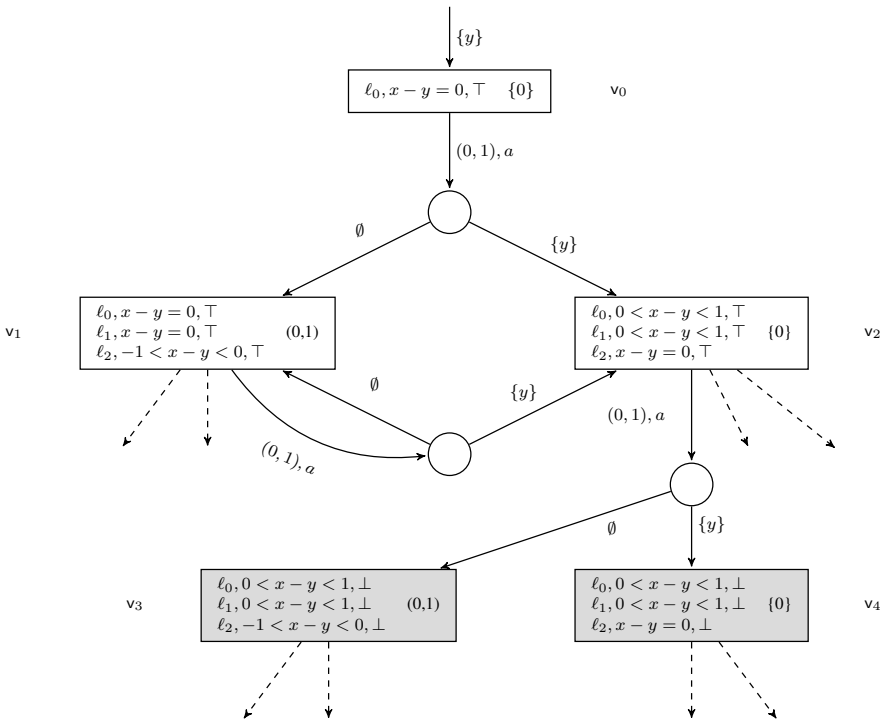


Figure 3.2: Excerpt the game $\mathcal{G}_{\mathcal{A},(1,1)}$.

configurations are $\top$ because the guard $0 < x < 1$ has not been approximated. Last, the region on $\{y\}$ of v_1 is naturally $0 < y < 1$.

In the state v_2 obtained when y is reset by Determinizator, the relations differ. For the configuration with location ℓ_2 the relation is simply $x - y = 0$ since both x and y were reset. The two other configurations share the relation $0 < x - y < 1$ derived from $0 < x < 1$ and y reset. In v_2 again all the booleans are true and the associated region is $\{0\}$. Note that from v_1, the move $(0 < y < 1, a)$ of Spoiler yields exactly the same successors as from v_0. Indeed, the only relevant configuration when firing action a is the one with location ℓ_0 (because there no a-transition can be fired from ℓ_1 or ℓ_2) and the configuration associated with ℓ_0 in v_1 is exactly the same as the unique configuration in v_0.

We end this example by detailing the construction of the successors for v_2, assuming Spoiler chose the move $(0 < y < 1, a)$. Here also the only relevant configuration in v_2 is $(\ell_0, 0 < x - y < 1, \top)$, because there are no a transitions in $\mathcal{A}$ from ℓ_1 and ℓ_2. Since the relation $0 < x - y < 1$ is different from $x - y = 0$, the guard on x induced by the region $0 < y < 1$ is not trivial. Figure 3.3 illustrates the computation of the guard over x induced by the relation $C = 0 < x - y < 1$ and the region $r' = 0 < y < 1$. The dotted area represents the set of valuations over $\{x, y\}$ which satisfy $0 < y < 1$, and the dashed area represents the relation $C = 0 < x - y < 1$ with the maximal constant 1. The induced guard $[r' \cap C]_{|\{x\}}$ (*i.e.* the guard over x encoded by the guard r' on y through the relation C) is then the projection over clock x of the intersection of these two areas. In this example, the induced guard is $0 < x < 2$. In fact, the figure represents the computation of the induced guard, without taking into account the maximal constant. Since the maximal constant is one, the induced guard is not $0 < x < 2$: it is $0 < x$. Therefore, the transitions of $\mathcal{A}$ corresponding to the choice of Spoiler $(0 < y < 1, a)$ are the three possible transitions with source ℓ_0, but they are over-approximated.

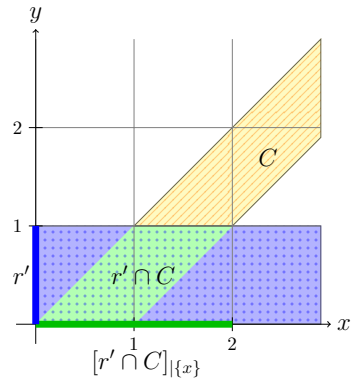


Figure 3.3: Construction of the induced guard.

Indeed, the induced guard $[r' \cap C]_{|\{x\}} = 0 < x$ is not included in the original guard $g = 0 < x < 1$ in $\mathcal{A}$, *i.e.*, the region r' possibly encodes more values than the guard g. As a consequence, all the configurations in v_3 and v_4 are marked $\perp$, and thus these states belong to Bad, represented by the gray color in Figure 3.2.

It remains to detail the computation of the relations in v_3 and v_4. Assuming Determinizator chooses not to reset y leads to v_3, in which for the configuration with location ℓ_0, the relation is the smallest one containing $(0 < x-y < 1) \cap (0 < y < 1) \cap (0 < x < 1)$, that is $0 < x-y < 1$. The relation for the last configuration is $\overleftarrow{\big((0 < x - y < 1) \cap (0 < y < 1) \cap (0 < x < 1)\big)}_{[x \leftarrow 0]}$, which is same as $\overleftarrow{(x = 0 < y < 1)}$, that is $-1 < x - y < 0$.

The complete construction of the game $\mathcal{G}_{\mathcal{A},(1,1)}$ is depicted in Figure 3.4, together with a winning strategy for Determinizator represented by bold edges.

3.1.3 Properties of the strategies

Given $\mathcal{A}$ a timed automaton and resources (k, M'), the game $\mathcal{G}_{\mathcal{A},(k,M')}$ is a finite-state safety game for Determinizator. The possible behaviors in the game are expressed by means of strategies. Intuitively, a strategy for player Determinizator (resp. Spoiler) chooses which move to perform from vertex $\mathsf{v}_D \in \mathsf{V}_D$ (resp. $\mathsf{v}_S \in \mathsf{V}_S$) based on the history of the game so far. It is a classical result that, for safety games, winning strategies can be chosen positional (the chosen move only depends on the current vertex) and they can be computed in linear time in the size of the arena [GTW02]. Therefore, in the following, we only consider positional strategies, and simply write "strategies" for "positional strategies".

A strategy for player Determinizator is thus described by a function $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$ assigning to each state $\mathsf{v}_D \in \mathsf{V}_D$ a set $Y' \subseteq Y$ of clocks to be reset; the successor state is then $\mathsf{v}_S \in \mathsf{V}_S$ such that $(\mathsf{v}_D, Y', \mathsf{v}_S) \in \delta$. Symmetrically, a strategy for Spoiler is a mapping $\sigma' : \mathsf{V}_S \to \mathsf{Act}_S$ assigning to each state $\mathsf{v}_S \in \mathsf{V}_S$ a region over Y and an action $a \in \Sigma$; the successor state is then $\mathsf{v}_D \in \mathsf{V}_D$ such that $(\mathsf{v}_S, (r', a), \mathsf{v}_D) \in \delta$. A pair of strategies (σ, σ') (one for each of the players) yields a path, written $\pi_{\sigma,\sigma'}$ in the game graph, which is finite or has a lasso shape: a prefix path starting from v_0 followed by a cycle. A strategy σ for Determinizator is *winning* if whatever the strategy σ' for Spoiler, the path $\pi_{\sigma,\sigma'}$ does not visit any Bad states.

With every strategy for Determinizator σ we associate the timed automaton $\mathsf{Aut}(\sigma)$ obtained by

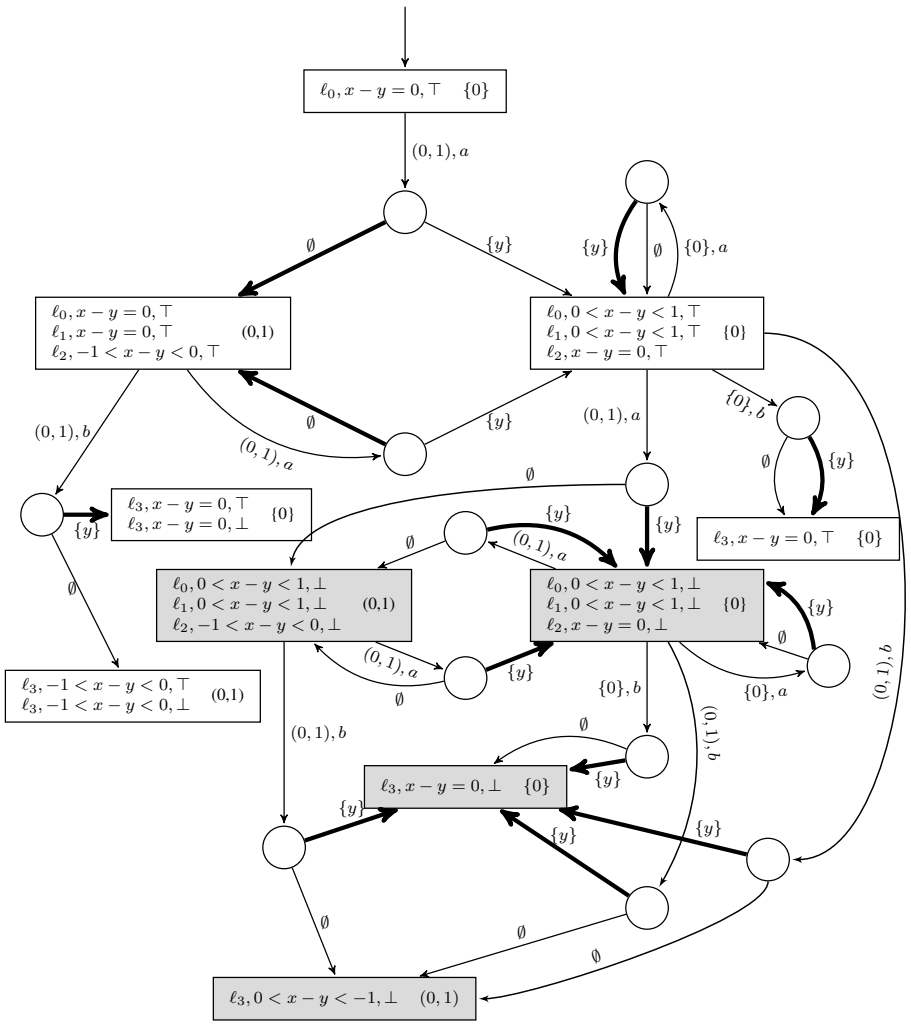


Figure 3.4: The game $\mathcal{G}_{\mathcal{A},(1,1)}$ and an example of winning strategy σ for Determinizator.

merging a transition of Spoiler with the transition chosen by Determinizator just after, and setting final locations as states of Spoiler containing at least one final location of $\mathcal{A}$.

Definition 3.2. *Let $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ be the game built from a timed automaton $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ and resources (k, M'). With a strategy for Determinizator $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$, we associate the timed automaton $\mathsf{Aut}(\sigma) = (\mathsf{V}_S, \mathsf{v}_0, F', \Sigma, Y, M', E')$ defined by:*

- *V_S is the set of locations, with v_0 the initial location;*
- *$F' = \{\mathsf{v}_S = (\mathcal{E}, r) \in \mathsf{V}_S \mid \exists(\ell, C, b)$ with $\ell \in F\}$ is the set of final locations;*
- *Y is the set of k clocks used in $\mathcal{G}_{\mathcal{A},(k,M')}$;*
- *the set of edges is $E' = \{(\mathsf{v}_S, g, a, Y', \mathsf{v}'_S) \in \mathsf{V}_S \times G_{M'}(Y) \times \Sigma \times 2^Y \times \mathsf{V}_S \mid \exists \mathsf{v}_D \in \mathsf{V}_D$ such that $(\mathsf{v}_S, (g, a), \mathsf{v}_D) \in \delta$, $(\mathsf{v}_D, Y', \mathsf{v}'_S) \in \delta$ and $\sigma(\mathsf{v}_D) = Y'\}$.*

The main result of the chapter is stated in the following theorem and links strategies of Determinizator with deterministic over-approximations of the initial timed language.

Theorem 3.1. *Let $\mathcal{A}$ be a timed automaton with no ε-transition and no invariant, and (k, M') resources. For every strategy σ of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$, $\mathsf{Aut}(\sigma)$ is a deterministic timed automaton over resources (k, M') and satisfies $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$. Moreover, if σ is winning, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$.*

The full proof is given in the general case with ε-transitions and invariants in $\mathcal{A}$ in Section 3.3.3; we however give below the main ideas for this simpler case.

Sketch. Given a strategy σ for Determinizator, we show that there exists a weak timed simulation between $\mathcal{A}$ and $\mathsf{Aut}(\sigma)$, namely the relation $\mathcal{R}$ defined by: $\mathcal{R} = \{((\ell, v), ((\mathcal{E}, r), v')) \mid \exists (\ell, C, b) \in \mathcal{E},\ (v, v') \in C \wedge v' \in \overrightarrow{r}\}$. This entails the language inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$.

Assuming now that σ is winning, given a run ϱ in $\mathsf{Aut}(\sigma)$, one can build backwards a path in $\mathcal{A}$, from an accepting configuration to the initial one applying elementary predecessors. Since σ is winning, guards are not over-approximated in $\mathsf{Aut}(\sigma)$, and there is a run in $\mathcal{A}$ with the same delays as in ϱ and following the path. This entails the reverse language inclusion $\mathcal{L}(\mathsf{Aut}(\sigma)) \subseteq \mathcal{L}(\mathcal{A})$. □

Standard techniques based on the computation of attractors allow one to check for the existence of a winning strategy for Determinizator, and in the positive case, to extract such a strategy [GTW02]. Our game construction can thus be applied to construct deterministic equivalent (or deterministic over-approximations) to timed automata. On our running example, on Figure 3.4, a winning strategy σ_{Win} for Determinizator is represented by the bold edges. This strategy yields the deterministic equivalent for $\mathsf{Aut}(\sigma_{\mathsf{Win}})$ depicted in Figure 3.5.

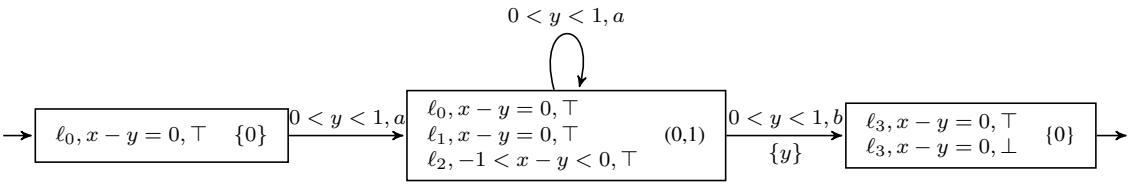


Figure 3.5: The deterministic TA $\mathsf{Aut}(\sigma_{\mathsf{Win}})$ obtained by our construction.

Remark 3.1. *In the approach for the diagnosability problem [BCD05] from which our game construction is inspired, the existence of a winning strategy is equivalent to the existence of a diagnoser with given resources. In comparison, recall that the determinizability problem with fixed resources is undecidable [Tri06, Fin06]. As a consequence, in our context, there is no hope to have a reciprocal statement to the one of Theorem 3.1. In particular, assuming that $\mathcal{A}$ can be determinized with resources (k, M') does not imply that Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(k,M')}$. To illustrate this phenomenon, Figure 3.6 represents a timed automaton $\mathcal{A}$ which is determinizable with resources $(1, 1)$, but for which Determinizator has no winning strategy in $\mathcal{G}_{\mathcal{A},(1,1)}$. Intuitively the self loop on ℓ_0 forces Determinizator to reset the clock in its first move; afterwards, on each branch of the automaton (via ℓ_1, ℓ_2 or ℓ_3), the behavior of $\mathcal{A}$ is strictly over-approximated in the game. However, each over-approximation on a branch is "covered" by the other branches, so that the losing strategy yields a deterministic equivalent to $\mathcal{A}$, represented on Figure 3.7.*

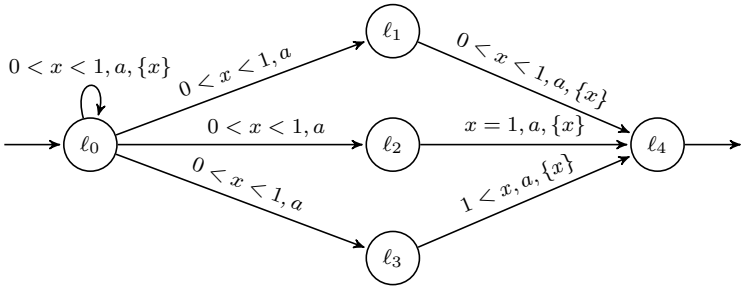


Figure 3.6: A determinizable TA for which there is no winning strategy for Determinizator.

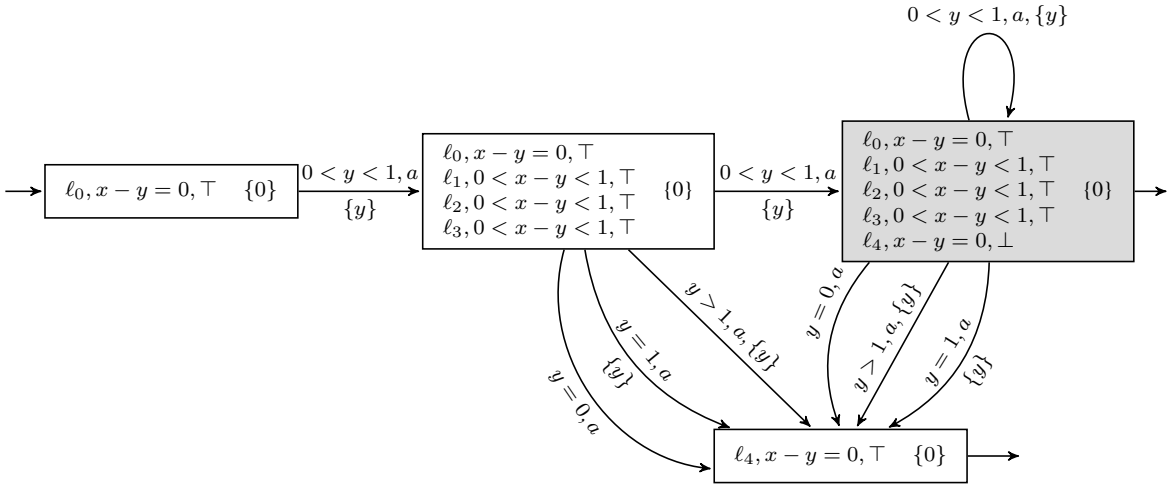


Figure 3.7: A deterministic equivalent to the TA in Figure 3.6 obtained with a losing strategy.

Remark 3.2. *The size of the game is doubly exponential in the size of the original timed automaton and we do not have a better upper bound for the resulting deterministic timed automaton. Note that any deterministic timed automaton* $\mathsf{Aut}(\sigma)$ *has diagonal guards, since its transitions are guarded by regions over* Y*. Yet, this diagonal guards can be removed avoiding the traditional exponential blowup, because the satisfaction of diagonal constraints is already encoded in the region associated with each location. From a timed automaton without diagonal guards, our method thus allows one to construct a deterministic over-approximation without diagonal guards.*

Atomic resets We now establish that winning strategies for Determinizator can be chosen in the restricted class of positional strategies with atomic resets. A strategy σ for Determinizator is with atomic resets if for every move $Y' \subseteq Y$ in σ, $|Y'| \leq 1$: in words, at most one clock is reset on each move of Determinizator.

Proposition 3.1. *Determinizator has a winning strategy* $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$ *if and only if it has a winning strategy with atomic resets* $\sigma' : \mathsf{V}_D \to Y \cup \{\emptyset\}$.

Proof. The proof only treats the direct implication because the other is trivial. Intuitively two clocks of Y with the same value do not give more information than a single one, so that it is never worth

resetting two or more clocks. More precisely, any timed automaton can be turned into a weakly timed bisimilar one with atomic resets only, using a construction similar to the one that removes clock transfers (*i.e.*, updates of the form $x := x'$) [BDFP04].

Let us assume that Y is totally ordered, and for $Y' \subseteq Y$, we write $\min(Y')$ for the smallest element in Y' according to the total order, with the convention that $\min(\emptyset) = \emptyset$. Given a winning strategy $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$ for Determinizator, we define $\sigma' : \mathsf{V}_D \to Y \cup \{\emptyset\}$, with atomic resets, iteratively. The idea is to simulate σ using only atomic resets. Instead of having several clocks with same value, we reset only one clock and use a mapping, along the construction, to store which clock is used to represent which set of clocks. Then, one considers triples with a state reached following σ', the corresponding state reached following σ and a mapping γ which assigns to each clock, the clock used in the state of σ' to represents its value in the state of σ.

- $\mathsf{Temp} := \{(\mathsf{v}, \mathsf{v}, \mathsf{Id}) \in \mathsf{V}_D \times \mathsf{V}_D \times Y^Y \mid \exists (r, a) \in \mathsf{Act}_S \text{ s.t. } (\mathsf{v}_0, r, a, \mathsf{v}) \in \delta\}$
- $\mathsf{V}_{\mathsf{def}} := \emptyset$
- While $\mathsf{Temp} \neq \emptyset$
  - take $(\mathsf{v}', \mathsf{v}, \gamma)$ in Temp
  - if $\mathsf{v}' \notin \mathsf{V}_{\mathsf{def}}$ then
    - $\sigma'(\mathsf{v}') := \min(\sigma(\mathsf{v}))$
    - $\gamma'(y) := \begin{cases} \sigma'(\mathsf{v}') & \text{if } y \in \sigma(\mathsf{v}), \\ \gamma(y) & \text{otherwise} \end{cases}$
    - $\mathsf{Temp} := \mathsf{Temp} \cup \{(\mathsf{w}', \mathsf{w}, \gamma') \mid \exists \mathsf{v}'_S, \mathsf{v}_S \in \mathsf{V}_S,\ \exists r'' \in \mathsf{Reg}^Y_{M'},\ \exists b \in \Sigma \text{ s.t. } \mathsf{v}' \xrightarrow{\sigma'(\mathsf{v}')} \mathsf{v}'_S \xrightarrow{(r'', b)} \mathsf{w}' \ \wedge\ \mathsf{v} \xrightarrow{\sigma(\mathsf{v})} \mathsf{v}_S \xrightarrow{(r''_{[y \leftarrow \gamma'(y)]}, b)} \mathsf{w} \}$
      Note that by definition of the set of edges of the game, $\mathsf{w} = (\mathsf{v}_S, (r''_{[y \leftarrow \gamma'(y)]}, b))$ and $\mathsf{w}' = (\mathsf{v}'_S, (r'', b))$.
    - $\mathsf{V}_{\mathsf{def}} := \mathsf{V}_{\mathsf{def}} \cup \{\mathsf{v}'\}$
- For all $\mathsf{v}_D \in \mathsf{V}_D \setminus \mathsf{V}_{\mathsf{def}}$
  - $\sigma'(\mathsf{v}_D) = \emptyset$

Intuitively, the above algorithm is a traversal of $\mathsf{Aut}(\sigma)$. We propagate the encoding of the clocks of σ by clocks in σ' and iteratively build σ'. The set Temp represents the triples to be processed and the set $\mathsf{V}_{\mathsf{def}}$ represents the states of Determinizator in the game for which the strategy σ' is defined. Moreover, the last step of the algorithm consists in arbitrarily defining the strategy for the unvisited states. By construction, these states are not reachable when Determinizator follows σ' hence this choice does not impact. Thus, σ' is well defined. The correction is a bit tedious, but intuitive: relations in the states in σ' give at least as much information as in σ because the time information of each clock x in σ is carried by $\gamma(x)$ in the corresponding state in σ'. The duplication of the information does not help to win.

Let us prove formally that σ' is a winning strategy using the following lemma.

Lemma A. *For all* $(\mathsf{v}', \mathsf{v}, \gamma) \in \mathsf{V}_{\mathsf{def}}$ *with* $\mathsf{v}' = (\mathcal{E}_{\mathsf{v}'}, (r', a'))$ *and* $\mathsf{v} = (\mathcal{E}_{\mathsf{v}}, (r, a))$*:*

i) $a' = a$

ii) $r = r'_{[y \leftarrow \gamma(y)]}$

iii) v *has no predecessor in* Bad

iv) $\forall \mathsf{w}'$ *such that* $\exists (\mathsf{v}'_S, r'', b) \in \mathsf{V}_S \times \mathsf{Act}_S$ *with* $\mathsf{v}' \xrightarrow{\sigma'(\mathsf{v}')} \mathsf{v}'_S \xrightarrow{(r'',b)} \mathsf{w}'$*, then* $\exists (\mathsf{w}_0, \gamma_0) \in \mathsf{V}_D \times Y^Y$ *such that* $(\mathsf{w}', \mathsf{w}_0, \gamma_0) \in \mathsf{V}_{\mathsf{def}}$

v) $\mathcal{E}_{\mathsf{v}} \subseteq \{(\ell, C'_{[y \leftarrow \gamma(y)]}, \top) \mid (\ell, C', \top) \in \mathcal{E}_{\mathsf{v}'}\} \cup \{(\ell, C, \bot)\}$

vi) $\mathcal{E}_{\mathsf{v}'} \subseteq \{(\ell, C', b') \mid (\ell, C'_{[y \leftarrow \gamma(y)]}, b) \in \mathcal{E}_{\mathsf{v}}\}$

The proof of Lemma A is simple but tedious. Let us first assume that this lemma is true. The fifth item implies that for all $(\mathsf{v}', \mathsf{v}, \gamma) \in \mathsf{V}_{\mathsf{def}}$, $(\mathsf{v} \notin \mathsf{Bad} \Rightarrow \mathsf{v}' \notin \mathsf{Bad})$. The third item implies that the v''s are not in Bad. The fourth item implies that only these states v' (appearing in $\mathsf{V}_{\mathsf{def}}$) impact on the fact that σ' is winning or not. As a consequence, if Lemma A is true, then σ' is winning. □

Let us now prove Lemma A.

Proof of Lemma A. Remark that all triples in $\mathsf{V}_{\mathsf{def}}$ are first added to Temp. Items $i)$, $ii)$, $iii)$ are satisfied by triples added to Temp at the initialization. Moreover, the updates of Temp only add triples satisfying $i)$ and $ii)$ and whose a predecessor v_S of the second element is a state of Spoiler reachable following σ, hence $iii)$ is satisfied too (only the markers of the configurations of v_S impact on whether $\mathsf{v}_S \in \mathsf{Bad}$ or not, and all the predecessors of v share the same markers over the configurations, by definition of δ). Therefore, items $i)$, $ii)$, $iii)$ are satisfied for all triples of Temp and thus of $\mathsf{V}_{\mathsf{def}}$.

Let us prove by induction that items $v)$ and $vi)$ are satisfied for all triples $(\mathsf{v}', \mathsf{v}, \gamma) \in \mathsf{V}_{\mathsf{def}}$. Triples added to Temp at the initialization are such that $\mathcal{E}_{\mathsf{v}} = \mathcal{E}_{\mathsf{v}'}$, hence they satisfy $v)$ and $vi)$. Let us prove now that if $(\mathsf{v}', \mathsf{v}, \gamma) \in \mathsf{V}_{\mathsf{def}}$ satisfies $v)$ and $vi)$, then triples added to Temp during the inner loop for this triple, satisfy $v)$ and $vi)$ too. Let $(\mathsf{w}', \mathsf{w}, \gamma')$ with $\mathsf{w}' = (\mathcal{E}_{\mathsf{w}'}, (r'', b))$ and $\mathsf{w} == (\mathcal{E}_{\mathsf{w}}, (r''_{[y \leftarrow \gamma(y)]}, b))$ be any triple added to Temp from $(\mathsf{v}', \mathsf{v}, \gamma)$.

v) Let us first prove that $\mathcal{E}_{\mathsf{w}} \subseteq \{(\ell, C'_{[y \leftarrow \gamma'(y)]}, \top) \mid (\ell, C', \top) \in \mathcal{E}_{\mathsf{w}'}\} \cup \{(\ell, C, \bot)\}$. For any $(\ell, C, \top) \in \mathcal{E}_{\mathsf{w}}$, there exists $(\ell_0, C_0, \top) \in \mathcal{E}_{\mathsf{v}}$ such that by definition of the game:

- $\exists \ell_0 \xrightarrow{g,a,X'} \ell \in E$ s. t. $[r'_{[y \leftarrow \gamma(y)]} \cap C_0]_{|X} \cap g \neq \emptyset$
- $C = \overleftarrow{(r'_{[y \leftarrow \gamma(y)]} \cap C_0 \cap g)_{[X' \leftarrow 0][\sigma(\mathsf{v}) \leftarrow 0]}}$
- $[r'_{[y \leftarrow \gamma(y)]} \cap C_0]_{|X} \subseteq g$

By induction hypothesis, there exists $(\ell_0, C'_0, \top) \in \mathcal{E}_{v'}$ such that $C'_{0[y \leftarrow \gamma(y)]} = C_0$, then:

- $[r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]}]_{|X} \cap g \neq \emptyset$
- $[r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]}]_{|X} \subseteq g$

This implies that:

- $[r' \cap C'_0]_{|X} \cap g \neq \emptyset$
- $[r' \cap C'_0]_{|X} \subseteq g$

Indeed, $[r' \cap C'_0]_{|X} \subseteq [r'_{|Im\gamma \cup X} \cap C'_{0|Im\gamma \cup X}]_{|X} = [r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]}]_{|X}$. Then $[r'' \cap C'_0]_{|X} \cap g = \emptyset$ implies that $[r' \cap C'_0]_{|X} = \emptyset$ which is not possible if v' is a state of the game. As a consequence, there is a configuration $(\ell, C', \top)$ in $\mathcal{E}_{\mathsf{w}'}$ such that $C' = \overleftarrow{(r' \cap C'_0 \cap g)_{[X' \leftarrow 0][\sigma'(\mathsf{v}') \leftarrow 0]}}$. Then:

$$\begin{array}{rcl} C'_{[y \leftarrow \gamma'(y)]} & = & \overleftarrow{(r' \cap C'_0 \cap g)_{[X' \leftarrow 0][\sigma'(\mathsf{v}') \leftarrow 0][y \leftarrow \gamma'(y)]}} \\ & = & \overleftarrow{(r' \cap C'_0 \cap g)_{[X' \leftarrow 0][\sigma'(\mathsf{v}') \leftarrow 0][y \leftarrow \gamma(y)][\sigma(\mathsf{v}) \leftarrow \sigma'(\mathsf{v}')]}} \\ & = & \overleftarrow{(r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]} \cap g)_{[X' \leftarrow 0][\sigma(\mathsf{v}) \leftarrow \sigma'(\mathsf{v}')]}} \\ & = & \overleftarrow{(r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]} \cap g)_{[X' \leftarrow 0][\sigma(\mathsf{v}) \leftarrow 0]}}. \end{array}$$

Therefore, $(\mathsf{w}', \mathsf{w}, \gamma')$ satisfies $v)$. Finally, all the triples in $\mathsf{V}_{\mathsf{def}}$ are added in Temp first, so that by induction $v)$ is satisfied by all the triples in $\mathsf{V}_{\mathsf{def}}$.

$vi)$ Let us now prove that $\mathcal{E}_{\mathsf{w}'} \subseteq \{(\ell, C', b') \mid (\ell, C'_{[y \leftarrow \gamma(y)]}, b) \in \mathcal{E}_{\mathsf{w}}\}$. For any $(\ell, C', b') \in \mathcal{E}_{\mathsf{w}'}$, there exists $(\ell_0, C'_0, b) \in \mathcal{E}_{\mathsf{v}'}$ such that:

- $\exists \ell_0 \xrightarrow{g,a,X'} \ell \in E$ s. t. $[r' \cap C'_0]_{|X} \cap g \neq \emptyset$
- $C = \overleftarrow{(r' \cap C_0 \cap g)_{[X' \leftarrow 0][\sigma'(\mathsf{v}') \leftarrow 0]}}$

By induction hypothesis, there exists $(\ell_0, C'_{0[y \leftarrow \gamma(y)]}, b) \in \mathcal{E}_{\mathsf{v}}$, then:

- $\emptyset \neq [r' \cap C'_0]_{|X} \cap g \subseteq [r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]}]_{|X} \cap g$

As a consequence, there is a configuration (ℓ, C, b) in $\mathcal{E}_{\mathsf{w}}$ such that the relation C is equal to $\overleftarrow{(r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]} \cap g)_{[X' \leftarrow 0][\sigma(\mathsf{v}) \leftarrow 0]}}$, which has been proved to be equal to $C'_{[y \leftarrow \gamma'(y)]}$ above. Therefore $vi)$ is satisfied by $(\mathsf{w}', \mathsf{w}, \gamma')$ and, by induction, by all the triples in $\mathsf{V}_{\mathsf{def}}$.

Finally, let us prove that $iv)$ is also satisfied for all triples in $\mathsf{V}_{\mathsf{def}}$. Let $(\mathsf{v}', \mathsf{v}, \gamma)$ be a triple in $\mathsf{V}_{\mathsf{def}}$. Let $\mathsf{w}' = (\mathcal{E}_{\mathsf{w}'}, (r'', b))$ be a state of Determinizator such that $\exists (\mathsf{v}'_S, r'', b) \in \mathsf{V}_S \times \mathsf{Act}_S$ such that $(\mathsf{v}', \sigma'(\mathsf{v}'), \mathsf{v}'_S) \in \delta \ \wedge \ (\mathsf{v}'_S, (r'', b), \mathsf{w}') \in \delta$. Then, there exists $(\ell_0, C'_0, b'_0) \in \mathcal{E}_{\mathsf{w}'}$ such that:

- $\exists \ell_0 \xrightarrow{g,b,X'} \ell' \in E$ s. t. $[r'' \cap C'_0]_{|X} \cap g \neq \emptyset$

Moreover, there exists $(\ell_1, C'_1, b'_1) \in \mathcal{E}_{\mathsf{v}'}$ such that:

- $\exists \ell_1 \xrightarrow{g_1,a,X'_1} \ell_0 \in E$ s. t. $[r' \cap C'_1]_{|X} \cap g_1 \neq \emptyset$
- $C'_0 = \overleftarrow{(r' \cap C'_1 \cap g_1)_{[X' \leftarrow 0][\sigma'(\mathsf{v}') \leftarrow 0]}}$.

As a consequence there exists a configuration $(\ell, C'_{1[y \leftarrow \gamma(y)]}, b_1) \in \mathcal{E}_v$ such that:

- $\emptyset \neq [r' \cap C'_1]_{|X} \cap g_1 \subseteq [r'_{[y \leftarrow \gamma(y)]} \cap C'_{1[y \leftarrow \gamma(y)]}]_{|X} \cap g_1$.

Then there exists a configuration (ℓ_0, C_0, b_0) of the successor v_S of v by the reset $\sigma(\mathsf{v})$ such that:

- $C_0 = \overleftarrow{(r'_{[y \leftarrow \gamma(y)]} \cap C'_{1[y \leftarrow \gamma(y)]} \cap g_1)_{[X' \leftarrow 0][\sigma(\mathsf{v}) \leftarrow 0]}}$

If we define γ' as expected ($\gamma'(y) = \gamma(y)$ if $y \notin \sigma(\mathsf{v})$, $\gamma'(y) = \sigma'(\mathsf{v}')$ otherwise) then $C_0 = C'_{0[y\leftarrow\gamma'(y)]}$. Moreover, $\emptyset \neq [r'' \cap C'_0]_{|X} \cap g \subseteq [r''_{[y\leftarrow\gamma'(y)]} \cap C'_{0[y\leftarrow\gamma'(y)]}]_{|X} \cap g$. Hence $(r''_{[y\leftarrow\gamma'(y)]}, b)$ is a possible move of Spoiler from v_S. Therefore there exists $\mathsf{w} \in \mathsf{V}_D$ such that $(\mathsf{v}, \sigma(\mathsf{v}), \mathsf{v}_S) \in \delta$ and $(\mathsf{v}_S, (r''_{[y\leftarrow\gamma'(y)]}, b), \mathsf{w}) \in \delta$. As a consequence, the triple $(\mathsf{w}', \mathsf{w}, \gamma')$ is added to Temp, then when this triple is taken from Temp, either $\sigma'(\mathsf{w}')$ is not already defined and this triple is added to $\mathsf{V}_{\mathsf{def}}$ or $\sigma'(\mathsf{w}')$ has been already defined and another triple of the form $(\mathsf{w}', \mathsf{w}_0, \gamma_0)$ has been added in $\mathsf{V}_{\mathsf{def}}$. Therefore $iv)$ is satisfied by all triples $(\mathsf{v}', \mathsf{v}, \gamma)$ in $\mathsf{V}_{\mathsf{def}}$. □

We gave the definition of our game approach for the determinization of timed automata without invariants and ε-transitions. Before extending the procedure to a richer model, we compare, in Section 3.2 this first version to two existing approaches which were introduced for this class of timed automata.

3.2 Comparison both existing methods

In this section, we compare the game approach presented in Section 3.1 to the existing methods: on the one hand, the approximation determinization algorithm from [KT09] and on the other hand the determinization procedure from [BBBB09]. We argue that our approach is both more precise than the algorithm of [KT09] and more general than the procedure of [BBBB09].

3.2.1 Comparison with [KT09]

First of all, our method extends [KT09] since each time the latter algorithm produces a deterministic equivalent with resources (k, M') for a timed automaton $\mathcal{A}$, there is a winning strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$. To be more precise, in [KT09] the construction of a deterministic over-approximation is guided by a *skeleton*, a finite automaton which governs the clock resets in the deterministic timed automaton in construction. The resets are thus defined by a regular untimed language. Our strategies are more powerful than the skeletons of [KT09] since the resets also depend on the regions the actions are taken in. Strategies can thus be seen as timed skeletons (the resets are defined by a regular timed language with given resources) and the game allows us to choose a good strategy, contrary to the skeletons of [KT09] that are fixed *a priori*. Also, when a strategy is winning, we know that the determinization is exact, while there is no such criterion in the work of Krichen et al. [KT09].

Second, contrary to the algorithm presented in [KT09], our game-approach is exact on deterministic timed automata: given a DTA $\mathcal{A}$ over resources (k, M), Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(k,M)}$. This is again a consequence of the more general fact that a strategy can be seen as a timed generalization of the notion of skeleton, and solving our game amounts to finding a relevant timed skeleton. As an example, on the timed automaton of Figure 3.1, with resources $(1, 1)$ and resetting the single clock y after each action, the algorithm of [KT09] produces a strict over-approximation, represented on Figure 3.8, while our approach, with the same resources, is exact.

Last, our approach also improves the precision of the relations between clocks by taking the original guard into account when computing the updated relation. Precisely, an intersection with the guard in the original TA is performed during the computation of the update up. This easy modification refines the over-approximation given by [KT09], and thus our method would perform better than [KT09] even assuming the same simple resetting policies (i.e. assuming strategies correspond to regular untimed languages) as in the over-approximation algorithm.

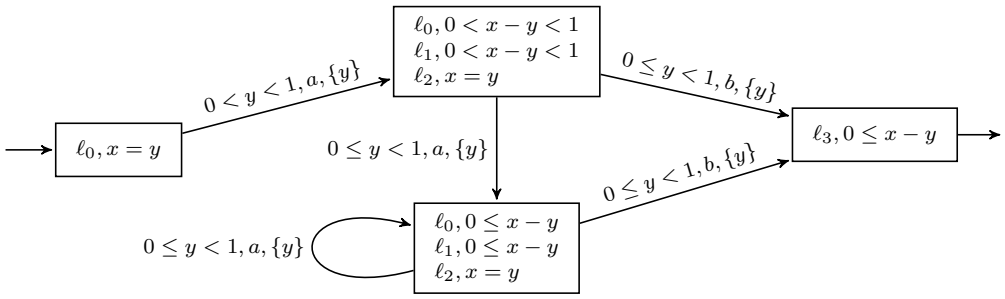


Figure 3.8: The result of algorithm [KT09] on the running example.

Inspired by the skeletons of [KT09], we explain how to naturally extend the class of event-clock timed automata into a class determinizable by our game-based approach. An *event-clock* automaton over alphabet Σ is a timed automaton with one clock per action which is reset exactly when the associated action is fired. The class of event-clock timed automata is determinizable [AFH94], and can be determinized by the procedure of [BBBB09]. In fact, event-clock automata enjoy a strong input-determinacy property: resets only depend on the untiming of the input word and neither on the timing information, nor on the run in the automaton. To extend the class while preserving determinizability, we introduce the notion of timed skeleton.

Definition 3.3. *Given a timed automaton $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$,* a timed skeleton *is a deterministic timed automaton $\mathsf{Sk} = (L^{\mathsf{Sk}}, \ell_0{}^{\mathsf{Sk}}, \emptyset, \Sigma, X, M, E^{\mathsf{Sk}})$ over the same resources (X, M) such that: Sk is complete and it governs the clock resets in $\mathcal{A}$, that is, in the synchronous product of $\mathcal{A}$ and Sk, clock resets of both automata coincide.*

This notion is inspired by the *skeleton* of Krichen and Tripakis [KT09], a deterministic finite automaton that guides the construction of a deterministic over-approximation by fixing the clock resets. It is also linked with input-determinacy, since if a timed automaton admits a timed skeleton, then the resets only depend on the input timed word and not the run.

Proposition 3.2. *Let $\mathcal{A}$ be a timed automaton with resources (X, M). If $\mathcal{A}$ admits a timed skeleton, then $\mathcal{A}$ is determinizable by our game-based approach with resources $(|X|, M)$.*

Proof. The idea of the proof is that the timed skeleton allows one to define a winning strategy for Determinizator. It is based on the fact that admitting a timed skeleton is equivalent to having resets dependent on the input timed word but not on the run.

Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ be a timed automaton that admits a timed skeleton $\mathsf{Sk} = (L^{\mathsf{Sk}}, \ell_0{}^{\mathsf{Sk}}, \emptyset, \Sigma, X, M, E^{\mathsf{Sk}})$. We consider the game $G = \mathcal{G}_{\mathcal{A},(|X|,M)}$ built from $\mathcal{A}$ with the same resources $(|X|, M)$. If Y is the set of new clocks, we write $\gamma : X \to Y$ for the bijection between X and Y.

Let us show that Sk induces a winning strategy for Determinizator: the strategy follows the resets from the skeleton. Formally, the strategy $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$ for Determinizator can be defined as follows:

- $\mathsf{Temp} := \{(\mathsf{v}_D, \ell_0{}^{\mathsf{Sk}}) \in \mathsf{V}_D \times L^{\mathsf{Sk}} \mid \exists (r, a) \in \mathsf{Act}_S \text{ s.t. } (\mathsf{v}_0, (r, a), \mathsf{v}_D) \in \delta\}$
- $\mathsf{V}_{\mathsf{def}} := \emptyset$
- While $\mathsf{Temp} \neq \emptyset$

  - pick and remove $(\mathsf{v}', \ell'^{\mathsf{Sk}})$ in Temp
  - if $\mathsf{v}' \notin \mathsf{V}_{\mathsf{def}}$ then
    * $\mathsf{V}_{\mathsf{def}} := \mathsf{V}_{\mathsf{def}} \cup \{\mathsf{v}'\}$
    * $\sigma(\mathsf{v}') := X'_{[x \leftarrow \gamma(x)]}$ where X' is the set of clocks reset along the edge outgoing from ℓ'^{Sk} in Sk with a guard $g \supseteq r_{[y \leftarrow \gamma^{-1}(y)]}$, and the action a such that $\mathsf{v}' = (\mathcal{E}', (r, a))$.
    * $\mathsf{Temp} := \mathsf{Temp} \cup \{(\mathsf{v}'', \ell''^{\mathsf{Sk}}) \mid \exists (\mathsf{v}'_S, r'', b) \in \mathsf{V}_S \times \mathsf{Act}_S \text{ s.t. } (\mathsf{v}', \sigma(\mathsf{v}'), \mathsf{v}'_S) \in \delta \wedge (\mathsf{v}'_S, (r'', b), \mathsf{v}'') \in \delta\}$

- For all $\mathsf{v}_D \in \mathsf{V}_D \setminus \mathsf{V}_{\mathsf{def}}$ such that $\sigma(\mathsf{v}_D)$ is not defined
  - $\sigma(\mathsf{v}_D) = \emptyset$

Intuitively, the strategy is defined performing the on-the-fly product with the skeleton, and simply following its reset policy through the mapping γ, which is simply a bijection between clocks of X and their copies in Y.

The correction of this construction is intuitive. It is based on the fact that, in every configuration of every location of the deterministic timed automaton resulting from strategy σ in G, the relation is included in $\wedge_{x \in X} x - \gamma(x) = 0$. Indeed, all the states of the resulting deterministic timed automaton are added in $\mathsf{V}_{\mathsf{def}}$, hence they are also added in Temp. All the states added in Temp at the initialization share the common relation $\wedge_{x \in X, y \in Y} x - y = 0$ which, in particular, contains $\wedge_{x \in X} x - \gamma(x) = 0$. Moreover, by definition of the skeleton, for each element $(\mathsf{v}', \ell'^{\mathsf{Sk}})$ in Temp, the resets of transitions from locations of configurations of v' are the same, and they are fixed by the resets of transitions from ℓ'^{Sk}. As the resets in σ are defined following the same skeleton up to the mapping γ, relations always contain $\wedge_{x \in X} x - \gamma(x) = 0$. As a consequence, there is no over-approximation of guard and the strategy σ is winning. □

The proof of Proposition 3.2 shows that there exists a winning strategy staying in states where relations are such that each clock in X is equal to a clock in Y. Hence, the result would also hold even if relations were restricted to mappings, and for the associated winning strategy, all the configurations have their boolean to $\top$. Moreover, if a timed automaton admits a timed skeleton, it can be determinized using the determinization procedure from [BBBB09].

3.2.2 Comparison with [BBBB09]

Our approach generalizes the one in [BBBB09] since, for any timed automaton $\mathcal{A}$ such that the procedure in [BBBB09] yields an equivalent deterministic timed automaton with k clocks and maximal constant M', there is a winning strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$. Intuitively this is a consequence of the fact that relations between clocks of $\mathcal{A}$ and clocks in the game generalize the mapping from [BBBB09], since a mapping can be seen as a restricted relation, namely a conjunction of constraints of the form $x - y = 0$.

Moreover, our approach strictly broadens the class of automata determinized by the procedure of [BBBB09] in two respects.

- First of all, our method allows one to cope with some language inclusions. For example, the TA depicted on the left-hand side of Figure 3.9 cannot be treated by the procedure of [BBBB09] but is easily determinized using our approach. In this example, the language of timed words

accepted in location ℓ_3 is not determinizable. This will cause the failure of [BBBB09]. However, all timed words accepted in ℓ_3 are also accepted in ℓ_4, and the language of timed words accepted in ℓ_4 is clearly determinizable. Our approach allows one to deal with such language inclusions thanks to the boolean ($\top$ or $\bot$) associated with each configuration, and will thus provide an equivalent deterministic timed automaton. This determinized version of the TA from Figure 3.9, left, was computed using a prototype implementation. We do not reproduce it here because it is quite large: it has 41 locations.

- Second, the relations between clocks of the TA and clocks of the game are more precise than the mappings used in [BBBB09]. For instance, the relation $x - y = 2$ suffices to express the value of a clock x thanks to a clock y; as another example, one can deduce that $x' < 2$ from $y' < 1$ assuming the relation $0 < x' - y' < 1$. The precision we add by considering relations rather than mappings is sometimes crucial for the determinization. For example, the TA represented on the right-hand side of Figure 3.9 can be determinized by our game-approach, but not by [BBBB09]. Intuitively, the loop of location ℓ_0 forces the procedure of [BBBB09] to introduce a new clock at each step of its unfolding, whereas the language remains the same if this loop is removed. A deterministic timed automaton obtained using our prototype using resources $(1, 1)$ for the TA from Figure 3.9, right, is depicted on Figure 3.10.

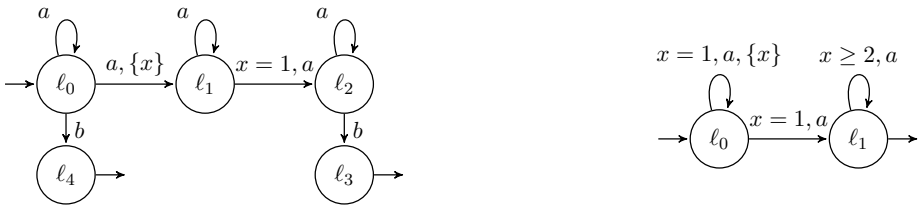


Figure 3.9: Examples of determinizable TA not treatable by [BBBB09].

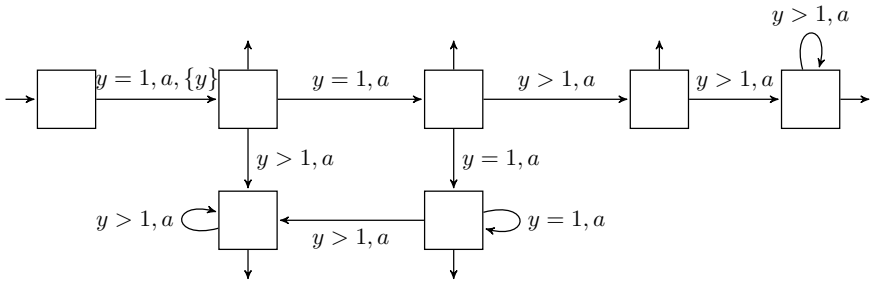


Figure 3.10: A deterministic equivalent of the TA of Figure 3.9, right.

Beyond broadening the class of timed automata that can be automatically determinized, our approach performs better on some timed automata by providing a deterministic timed automaton with less resources. This is the case on the running example of Figure 3.1. The deterministic automaton obtained by [BBBB09] is depicted in Figure 3.11: it needs 2 clocks when our method only needs one.

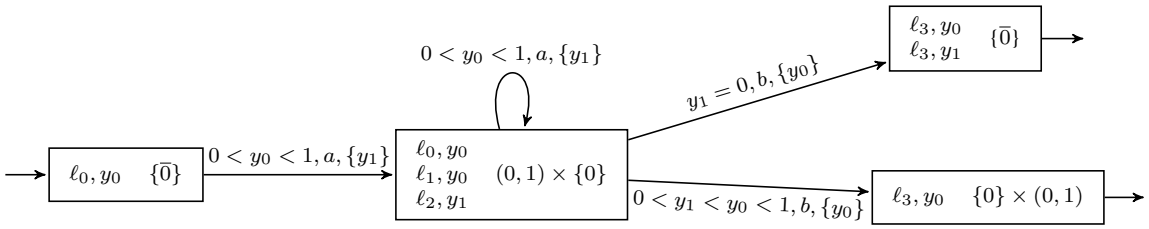


Figure 3.11: The result of procedure [BBBB09] on our running example.

The same phenomenon happens with timed automata with integer resets. Timed automata with integer resets, introduced in [SPKM08], form a determinizable subclass of timed automata, where every edge (ℓ, g, a, X', ℓ') satisfies $X' \neq \emptyset$ if and only if g contains an atomic constraint of the form $x = c$ for some clock x. Intuitively, a single clock is needed to represent clocks of $\mathcal{A}$ since they all share a common fractional part.

Proposition 3.3. *For every timed automaton $\mathcal{A}$ with integer resets and maximal constant M, Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(1,M)}$.*

Proof. Let $\mathcal{A}$ be a timed automaton with integer resets over set of clocks X and maximal constant M. Note that, by definition of TA with integer resets, along any run of $\mathcal{A}$, all clocks share the same fractional part. This crucial property ensures that an equivalent deterministic TA with one clock can be constructed. Precisely, in $\mathcal{G}_{\mathcal{A},(1,M)}$ we consider the strategy σ for Determinizator which resets the single clock y exactly for transitions that correspond to at least one transition of $\mathcal{A}$ containing an equality constraint (atomic constraint of the form $x = c$). Since $\mathcal{A}$ is a TA with integer resets, clocks in X cannot be reset out of these transitions. Therefore, for every clock $x \in X$, the value of y is always smaller than the one of x in $\mathsf{Aut}(\sigma)$ and each relation contains either $x - y = c$ with $0 \leq c \leq M$, or $x - y > M$. In the latter case, necessarily $x > M$. As a consequence, guards over X can always be exactly expressed in $G_M(\{y\})$. This ensures that only states where all configurations are marked $\top$ will be visited in $\mathsf{Aut}(\sigma)$. Hence, σ is winning and $\mathcal{L}(\mathsf{Aut}(\sigma)) = \mathcal{L}(\mathcal{A})$. Note that $\mathsf{Aut}(\sigma)$ is still a TA with integer resets and its size is doubly exponential in the size of $\mathcal{A}$. □

As a consequence of Proposition 3.3, any timed automaton with integer resets can be determinized into a doubly exponential single-clock timed automaton with the same maximal constant. This improves the result given in [BBBB09] where any timed automaton with integer resets and maximal constant M can be turned into a doubly exponential deterministic timed automaton, using $M + 1$ clocks. Moreover, our procedure is optimal on this class thanks to the lower-bound provided in [MK10]. Note also that the one-clock timed automaton we obtain coincides with the one obtained by the ad-hoc determinization of integer reset timed automata [MK10].

We discussed how our game approach improves the two existing methods for the determinization of timed automata. In the sequel, we define extensions of the game construction in order to deal with invariants and ε-transitions.

3.3 Extension to ε-transitions and invariants

In Section 3.1 the construction of the game and its properties were presented for a restricted class of timed automata with no ε-transitions and no invariants. Let us now explain how to extend the previous construction to deal with these two aspects.

3.3.1 ε-transitions

Let us first explain informally the modifications that are needed in the definition of the game to deal with ε-transitions.

Quite naturally, in order to remove ε-transitions, an ε-closure has to be performed when computing new states in the game. This closure calls for an extension of the structure of the states: delays might be mandatory before taking an ε-transition, and hence, potentially distinct regions are attached to configurations of a state in the game. This phenomenon is illustrated on the example of TA depicted in Figure 3.12, left. The resulting game, is represented Figure 3.12, right, for resources $(1, 2)$. There, the rightmost state is composed of the state reached without ε-transitions $\{(\ell_1, x - y = 0, \top, \{0\})\}$, and its ε-closure. For instance, the configuration $(\ell_1, x - y = -2, \top)$ can only be reached after two ε-transitions of the original TA, taken respectively after one and two time units. Thus this configuration can only happen when in clock y reaches region $\{2\}$ or later, whereas the configuration $(\ell_1, x - y = 0, \top)$ could be observed already in region $\{0\}$. As a consequence, within a state, configurations can have different associated regions, each region being a time-successor of the initial region of the state.

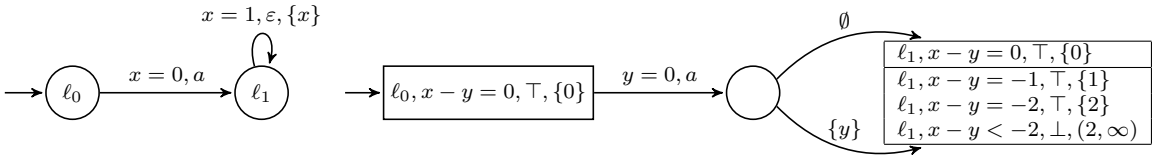


Figure 3.12: A timed automaton with ε-transitions and the resulting ε-closure.

Computing the ε-closure of a state in the game amounts to computing the set of reachable configurations by ε-transitions, and associating with every new configuration its corresponding region. This computation can be seen as a construction of a branch of the game where ε would be a standard action, but where Determinizator is not allowed to reset any clock; all the states obtained this way are then gathered into a unique state. For instance, Figure 3.13 represents the computation of the ε-closure discussed above, for a single clock y and maximum constant 2. This alternative point of view justifies that the computation always terminates.

Apart from the structure of the individual states, the set Bad also needs to be redefined when taking into account possible ε-transitions, in particular because regions are now attached to configurations and no longer to states. There are two new situations where an over-approximation might have occurred. First, if the ε-closure leads to a configuration with a final location after a non-zero delay. Second, if the configurations associated with a final locations in the upper part of the state (*i.e.* bearing the initial region) are marked $\bot$. In both cases, the state will be declared as final in the timed automaton $\mathsf{Aut}(\sigma)$ for any fixed strategy σ. However, Determinizator needs to avoid these states in order to ensure that no over-approximation occurred. Therefore, in these two situations, the state is added to the set Bad.

We now come to the formal definition of the game.

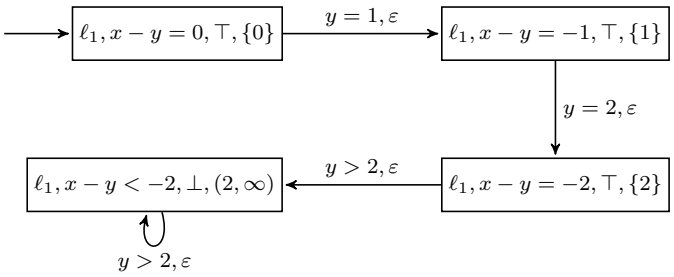


Figure 3.13: Step-wise computation of the ε-closure, before merging.

Definition 3.4. *Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ be a timed automaton and (k, M') resources. We let $\mathbf{M} = \max(M, M')$, and Y a set of k clocks. The game associated with $\mathcal{A}$ and (k, M') is $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ where:*

- $\mathsf{V} = \mathsf{V}_S \sqcup \mathsf{V}_D$ *is the finite set of vertices, with* $\mathsf{V}_S \subseteq 2^{L \times \mathsf{Rel}_{\mathbf{M}}(X \cup Y) \times \{\top,\bot\} \times \mathsf{Reg}^Y_{M'}}$ *and* $\mathsf{V}_D \subseteq \mathsf{V}_S \times \mathsf{Reg}^Y_{M'} \times \Sigma$*;*
- $\mathsf{v}_0 = \mathsf{cl}_\varepsilon(\{(\ell_0, \bigwedge_{z,z' \in X \cup Y} z - z' = 0, \top, \{\overline{0}\})\})$ *is the initial vertex, where* cl_ε *is defined in Equation* (3.5) *below, and* $\mathsf{v}_0 \in \mathsf{V}_S$*;*
- $\mathsf{Act} = \mathsf{Act}_S \sqcup \mathsf{Act}_D$ *is the set of possible actions,* $\mathsf{Act}_S = \mathsf{Reg}^Y_{M'} \times \Sigma$ *and* $\mathsf{Act}_D = 2^Y$*;*
- $\delta = \delta_S \cup \delta_D$ *is the transition relation with δ_S and δ_D defined as follows:*
  - $\delta_S \subseteq \mathsf{V}_S \times \mathsf{Act}_S \times \mathsf{V}_D$ *is the set of edges of the form* $\mathcal{E} \xrightarrow{(r',a)} (\mathcal{E}, (r', a))$ *if there exists* $(\ell, C, b, r) \in \mathcal{E}$ *such that* $r' \in \overrightarrow{r}$ *and there exists* $\ell \xrightarrow{g,a,X'} \ell' \in E$ *such that condition* $\mathsf{C}_{\neq\emptyset}$ *is satisfied, that is* $[r' \cap C]_{|X} \cap g \neq \emptyset$*,*
  - $\delta_D \subseteq \mathsf{V}_D \times \mathsf{Act}_D \times \mathsf{V}_S$ *is the set of edges of the form* $(\mathcal{E}, (r', a)) \xrightarrow{Y'} \mathcal{E}'$ *where*

$$\mathcal{E}' = \bigcup_{\gamma \in \mathcal{E}} \bigcup_{\gamma' \in \mathsf{Succ}^\varepsilon_e[r',a,Y'](\gamma)} \mathsf{cl}_\varepsilon(\gamma'); \tag{3.3}$$

    - *where* $\mathsf{Succ}^\varepsilon_e$ *is the elementary successor function*

$$\begin{aligned}\mathsf{Succ}^\varepsilon_e[r', a, Y'](\ell, C, b, r) = \{(\ell', C', b', r'_{[Y' \leftarrow 0]}) \mid \\ (\ell', C', b') \in \mathsf{Succ}_e[r', a, Y'](\ell, C, b)\},\end{aligned} \tag{3.4}$$

    - *and* cl_ε *is the ε-closure,* $\mathsf{cl}_\varepsilon(\ell, C, b, r)$ *is defined as the least fixpoint of the functional*

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{(\ell', C', b', r') \in X} \bigcup_{r'' \in \overrightarrow{r'}} \mathsf{Succ}^\varepsilon_e[r'', \varepsilon, \emptyset](\ell', C', b', r'), \tag{3.5}$$

- $\mathsf{Bad} = \big\{\{(\ell_j, C_j, \bot, r_j)\}_j\big\} \cup \big\{\{(\ell_j, C_j, b_j, r_j)\}_j \mid \forall h \ \big((\cup_j r_j \subseteq \overrightarrow{r_h}) \Rightarrow (\ell_h \in F \Rightarrow b_h = \bot)\big) \wedge (\exists i, \ \ell_i \in F)\big\}$ *is the set of bad states.*

We now detail the edge relation δ which gives the possible moves of the players. Given a state of Spoiler $\mathsf{v}_S = \{(\ell_j, C_j, b_j, r_j)_j\} \in \mathsf{V}_S$ and (r', a) one of his moves, the successor state is defined as the state $\mathsf{v}_D = (\{(\ell_j, C_j, b_j, r_j)\}_j, (r', a)) \in \mathsf{V}_D$ provided there exists $(\ell, C, b, r) \in \mathsf{v}_S$, such that $r' \in \overrightarrow{r}$ and there exists $\ell \xrightarrow{g,a,X'} \ell' \in E$ with $[r' \cap C]_{|X} \cap g \neq \emptyset$.

Given $\mathsf{v}_D = (\{(\ell_j, C_j, b_j, r_j)\}_j, (r', a)) \in \mathsf{V}_D$ a state of Determinizator and $Y' \subseteq Y$ one of his moves, the successor state v_S, formally defined above in Equation (3.3), is obtained as the ε-closure of the set of all elementary successors of configurations in $\{(\ell_j, C_j, b_j, r_j)\}_j$ by (r', a) and resetting Y'. Precisely, if (ℓ, C, b, r) is a configuration such that $r' \in \overrightarrow{r}$, its elementary successors set by (r', a) and resetting Y' are defined in Equation (3.4) using the basic elementary successor Succ_e defined in Equation (3.1) on page 44. To complete this definition, let us discuss the ε-closure of a state of Spoiler. The ε-closure of a configuration (ℓ, C, b, r), denoted by $\mathsf{cl}_\varepsilon(\ell, C, b, r)$, is the smallest set of configurations containing (ℓ, C, b, r) and closed under elementary successor for any pair (r', ε) where r' is a time successor of the source configuration and without resetting any clocks in Y. It is thus the least fixpoint of the functional in Equation (3.5). The termination in finite time of the iterative computation of the fixpoint comes from the fact that the number of configurations is finite.

3.3.2 Invariants

We now explain how to adapt the framework to timed automata with invariants. First, while computing the elementary successors for configurations, invariants have to be taken into account. Second, with each state of the game, we associate an invariant corresponding to the invariants of the original locations. Last, the set of bad states needs to be redefined.

An invariant over clocks of Y is attached to each state of Spoiler. A state v_S of Spoiler thus has the form $\mathsf{v}_S = (\{(\ell_j, C_j, b_j, r_j)\}_j, \mathsf{I})$ where $\mathsf{I} \in I_{M'}(Y)$ is intuitively the most restrictive invariant that over-approximates every invariant for the configurations composing v_S. Formally,

$$\mathsf{I} = \bigcup \big\{ r'' \in \mathsf{Reg}^Y_{M'} \mid \exists j,\ r'' \in \overrightarrow{r_j} \wedge [r'' \cap C_j]_{|X} \cap \mathsf{Inv}(\ell_j) \neq \emptyset \big\}. \tag{3.6}$$

In the computation of the successor states, the invariants are taken care of similarly to the guards: their satisfaction is checked on both end-points of the transitions. In order to do so, in the definition of $\mathsf{Succ}^\varepsilon_e$ for a transition $\ell \xrightarrow{g,a,X'} \ell'$ the condition $\mathsf{C}_{\neq\emptyset}$, that is $[r' \cap C]_{|X} \cap g \neq \emptyset$, is replaced with the condition $\mathsf{D}_{\neq\emptyset} := [r' \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell) \cap (\mathsf{Inv}(\ell'))_{[X' \leftarrow 0]^{-1}} \neq \emptyset$.

The boolean has to take into account the potential over-approximation of the invariant. It is thus redefined as follows:

$$b' = b \wedge \Big([r' \cap C]_{|X} \subseteq (g \cap \mathsf{Inv}(\ell) \cap (\mathsf{Inv}(\ell'))_{[X' \leftarrow 0]^{-1}}) \Big) \wedge \Big(\big[[\mathsf{Inv}(\ell) \cap C]_{|Y} \cap C \big]_{|X} = \mathsf{Inv}(\ell) \Big) \wedge \Big([\mathsf{Inv}(\ell) \cap C]_{|Y} = \mathsf{I} \Big) \tag{3.7}$$

where I is the invariant of the state containing this configuration. Indeed, in order to have no over-approximation of the invariant for a configuration, it is necessary that the invariant associated to the state of the game is not larger than the invariant induced by the configuration, moreover this latter has to not be an approximation of the invariant in $\mathcal{A}$. As a consequence, the configurations which are built via an approximation of some invariant are marked $\perp$. The relation update is also redefined, to enforce the satisfaction of the invariants:

$$\mathsf{up}(r', C, g, \ell, \ell', X', Y') = \overrightarrow{(r' \cap C \cap g \cap \mathsf{Inv}(\ell))_{[X' \leftarrow 0][Y' \leftarrow 0]} \cap \mathsf{Inv}(\ell')}. \tag{3.8}$$

Note that in this definition, the invariant and the configurations are interdependent. This is no problem. Configurations can be first computed assuming that the last condition $\left([\mathsf{Inv}(\ell) \cap C]_{|Y} = \mathsf{I}\right)$ for b' to be $\top$ is true. Then, I can be computed using configurations and last markers can be updated taking into account I.

Some over-approximation in the invariants can yield a strict over-approximation of the original timed language. The preservation of the property that any winning strategy for Determinizator yields a deterministic equivalent of the original timed automaton is ensured thanks to the booleans in configurations taking into account this risk. The definition of the set Bad is thus unchanged.

Let us illustrate the construction of the game, and in particular, the computation of invariants, on an example. Figure 4.1 represents a timed automaton with invariants. An excerpt of the corresponding

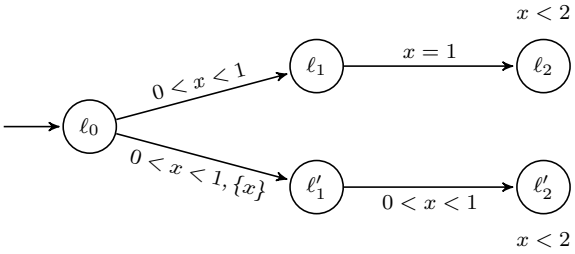


Figure 3.14: A timed automaton with invariants.

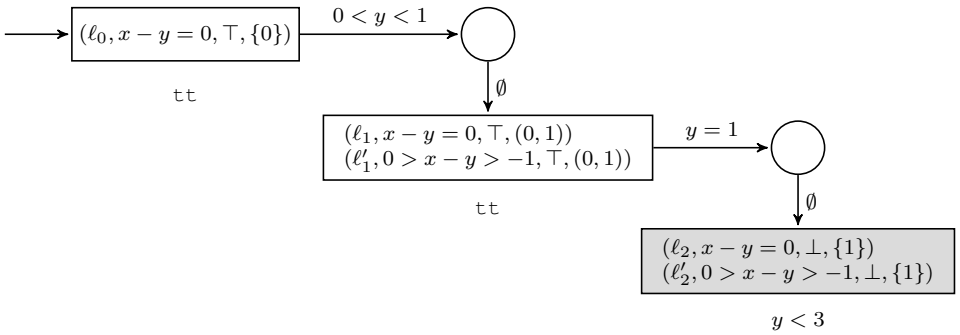


Figure 3.15: Excerpt of the game with resources $(1, 3)$ for the TA from Figure 4.1.

game, over resources $(1, 3)$, is depicted in Figure 3.15. We consider the right-most state in the picture, and explain how its configurations and its invariant are derived. For the first configuration, with location ℓ_2 and the induced invariant is $y < 2$, trivially obtained from the invariant $x < 2$ in ℓ_2 and the relation $x - y = 0$. The region over y is simply $y = 1$, and there was no over-approximations so far. The first configuration thus should be $(\ell_2, x - y = 0, \top, \{1\})$. Concerning the configuration with location ℓ'_2, the induced invariant is $y < 3$, since $x < 2$ and the relation $0 > x - y > -1$ imply $y < 3$. Note that the region $2 < y < 3$ is necessarily included in this invariant because *e.g.*, the valuation $x = 1.9$ and $y = 2.1$ satisfies $0 > x - y > -1$, $2 < y < 3$ and $x < 2$. Also, the boolean is $\bot$ since over-approximations occurred. The second configuration is thus $(\ell'_2, 0 > x - y > -1, \bot, \{1\})$. Last, the invariant associated with the state is the union of the invariants for each configurations $y < 2$ and $y < 3$. It is therefore over-approximated for the first configuration, which explains that its boolean is

set to $\perp$ in the end.

3.3.3 Properties of the strategies in the extended game

Theorem 3.1, for timed automata with no ε-transitions and no invariants, extends to timed automata with these features, using the extended game defined above. Recall that $\mathsf{Aut}(\sigma)$, the timed automaton derived from the game by fixing a strategy σ for Determinizator, is defined in Definition 3.2.

Theorem 3.2. *Let $\mathcal{A}$ be a timed automaton, and (k, M') resources. For every strategy σ of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$, $\mathsf{Aut}(\sigma)$ is a deterministic timed automaton over resources (k, M') and satisfies $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$. Moreover, if σ is winning, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$.*

Note that the game construction described in the current section is a conservative extension of the one given in Section 3.1.1 for timed automata with no ε-transitions and no invariants. As a consequence, the following proof of Theorem 3.2 also serves as a proof for Theorem 3.1.

Proof. The proof is split in two parts. First of all, we show that any strategy σ for Determinizator ensures $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$. Then we prove that the reverse inclusion also holds for every winning strategy σ.

($\subseteq$): Let σ be any strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$. To show that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$ we prove a stronger fact on the transition systems $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{\mathsf{Aut}(\sigma)}$ associated with $\mathcal{A}$ and $\mathsf{Aut}(\sigma)$: $\mathcal{T}_{\mathsf{Aut}(\sigma)}$ weak timed simulates $\mathcal{T}_{\mathcal{A}}$. Let $\mathcal{T}_{\mathcal{A}} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \rightarrow_{\mathcal{A}})$, $\mathcal{T}_{\mathsf{Aut}(\sigma)} = (\mathsf{S}', S'_0, \mathsf{S}'_F, (\mathbb{R}_+ \times \Sigma), \rightarrow_{\mathsf{Aut}(\sigma)})$, and $\mathcal{R} \subseteq S \times \mathsf{S}'$ the following binary relation:

$$\mathcal{R} = \left\{ \big((\ell, v), ((\mathcal{E}, \mathsf{I}), v')\big) \mid \exists (\ell, C, b, r) \in \mathcal{E},\ (v, v') \in C \wedge v' \in \overrightarrow{r} \right\}.$$

Let us prove that $\mathcal{R}$ satisfies the four conditions from Definition 2.3 on page 31, to be a weak timed simulation. Given that $\mathsf{Aut}(\sigma)$ has no ε-transitions, the fourth condition can be simplified. We will thus prove the following on $\mathcal{R}$:

(1) $(s_0, S'_0) \in \mathcal{R}$,

(2) $(s, S') \in \mathcal{R}$ and $s \in S_F$ implies $S' \in \mathsf{S}'_F$,

(3) for all $(s, S') \in \mathcal{R}$, for all $a \in \Sigma$ whenever $s \xrightarrow{a}_{\mathcal{A}} \tilde{s}$, there exists $\tilde{S}' \in \mathsf{S}'$ such that $(\tilde{s}, \tilde{S}') \in \mathcal{R}$ and $S' \xrightarrow{a}_{\mathsf{Aut}(\sigma)} \tilde{S}'$,

(4) for all $(s, S') \in \mathcal{R}$, whenever $s \xrightarrow{\tau_1}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} s_2 \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} s_n \xrightarrow{\tau_n}_{\mathcal{A}} \tilde{s}$, there exists $\tilde{S}' \in \mathsf{S}'$ such that $(\tilde{s}, \tilde{S}') \in \mathcal{R}$ and $S' \xrightarrow{\tau}_{\mathsf{Aut}(\sigma)} \tilde{S}'$ with $\tau = \sum_{i=1}^{n} \tau_i$.

(1) The first condition about the initial states is trivially satisfied, by definition of the initial state in the game, and thus the initial location in $\mathsf{Aut}(\sigma)$.

(2) Accepting locations in $\mathsf{Aut}(\sigma)$ are locations in which there is at least one configuration whose location is accepting in $\mathcal{A}$. As a consequence, the second condition is satisfied by $\mathcal{R}$.

Assume now that $s = (\ell_s, v_s)$ and $S' = ((\mathcal{E}_{S'}, \mathsf{I}_{S'}), v_{S'})$ are states of $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{\mathsf{Aut}(\sigma)}$ respectively such that $(s, S') \in \mathcal{R}$. Then, there exists a configuration $(\ell_s, C, b, r) \in \mathcal{E}_{S'}$ such that v_s and $v_{S'}$ satisfy the relation C, *i.e.* $(v_s, v_{S'}) \in C$, and $v_{S'} \in \overrightarrow{r}$. Moreover, if $v_s \in \mathsf{Inv}(\ell_s)$, which is true as soon as s is reachable from s_0, then $v_{S'} \in \mathsf{I}_{S'}$. Indeed, if $v_s \in \mathsf{Inv}(\ell_s)$, then the region $r_{S'}$ containing $v_{S'}$ is such that $r_{S'} \in \overrightarrow{r}$ and $(v_s, v_{S'}) \in r_{S'} \cap C$, hence the condition $\mathsf{D}_{\neq\emptyset}$ is satisfied because $v_s \in [r_{S'} \cap C]_{|X} \cap \mathsf{Inv}(\ell_s)$. Therefore, by definition of $\mathsf{I}_{S'}$ (see Equation (3.6)), $r_{S'} \subseteq \mathsf{I}_{S'}$ and thus $v_{S'} \in \mathsf{I}_{S'}$.

(3) Let us prove that the third condition is satisfied. Let $a \in \Sigma$ such that $s \xrightarrow{a}_{\mathcal{A}} \tilde{s}$. This transition comes from some edge $(\ell_s, g, a, X', \ell_{\tilde{s}})$ in $\mathcal{A}$ where $\ell_{\tilde{s}}$ is the location of $\tilde{s}$. Let $r_{S'}$ be the region containing $v_{S'}$. Then, $(v_s, v_{S'}) \in C$ implies that the condition $\mathsf{D}_{\neq\emptyset}$ is satisfied, because $v_s \in [r_{S'} \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell_s) \cap (\mathsf{Inv}(\ell_{\tilde{s}}))_{[X' \leftarrow 0]^{-1}}$. Hence, $\mathsf{Succ}_e^{\varepsilon}[r_{S'}, a, Y'](\ell_s, C, b, r)$ is not empty (whatever is $Y' \subseteq Y$). As a consequence, by definition of the game, there exists an edge $(\mathcal{E}_{S'}, \mathsf{I}_{S'}) \xrightarrow{r_{S'}, a, Y'} (\mathcal{E}_{\tilde{S}'}, \mathsf{I}_{\tilde{S}'})$ in $\mathsf{Aut}(\sigma)$, with some $Y' \subseteq Y$, and there exists a configuration $(\ell_{\tilde{s}}, C', b', r_{S'[Y' \leftarrow 0]}) \in \mathcal{E}_{\tilde{S}'}$ which is an elementary successor of (ℓ_s, C, b, r). Letting $\tilde{S}' = ((\mathcal{E}_{\tilde{S}'}, \mathsf{I}_{\tilde{S}'}), v_{\tilde{S}'})$ where $v_{\tilde{S}'} = v_{S'[Y' \leftarrow 0]}$, we observe that $(v_{\tilde{s}}, v_{\tilde{S}'}) \in C'$ using the definition of the updates for the relation (Equation (3.8) on page 61). Hence $(\tilde{s}, \tilde{S}') \in \mathcal{R}$, which proves condition (3) for $\mathcal{R}$.

(4) Finally, let us prove that $\mathcal{R}$ satisfies the last condition. Consider $s \xrightarrow{\tau_1}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} s_2 \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} s_n \xrightarrow{\tau_n}_{\mathcal{A}} \tilde{s}$, a sequence of delays and ε-transitions from s in $\mathcal{A}$. Letting $s_j = (\ell_j, v_j)$ (for $1 \leq j \leq n$) and $s = s_1$, for every $1 \leq j \leq n-1$ there exists an edge in $\mathcal{A}$ of the form $(\ell_j, g_j, \varepsilon, X_j, \ell_{j+1})$ with $v_j + \tau_j \models g_j \cap \mathsf{Inv}(\ell_j) \cap (\mathsf{Inv}(\ell_{j+1}))_{[X_j \leftarrow 0]^{-1}}$ and $v_{j+1} = (v_j + \tau_j)_{[X_j \leftarrow 0]}$. By definition of the ε-closure operator cl_ε (Equation (3.5) on page 60), for each index $1 \leq j \leq n$ there is a configuration $(\ell_j, C_j, b_j, r_j) \in \mathcal{E}_{s'}$ such that $(v_j, v_{S'} + \sum_{i=1}^{j-1} \tau_i) \in C_j$ and $v_{S'} + \sum_{i=1}^{j-1} \tau_i \in r_j$. As a consequence, $((\ell_j, v_j), ((\mathcal{E}_{S'}, \mathsf{I}_{S'}), v_{S'} + \sum_{i=1}^{j-1} \tau_i)) \in \mathcal{R}$ and, since the invariant $\mathsf{Inv}(\ell_j)$ is satisfied by v_j, we get that $v_{S'} + \sum_{i=1}^{j-1}$ satisfies $\mathsf{I}_{S'}$. In particular, this is true for $j = n$. Hence, $(v_n + \tau_n, v_{S'} + \sum_{i=1}^{n} \tau_i) \in C_n$ and $v_{\tilde{s}} = v_n + \tau_n$. Then, letting $\tilde{S}' = ((\mathcal{E}_{\tilde{S}'}, \mathsf{I}_{\tilde{S}'}), v_{S'} + \sum_{i=1}^{n-1} \tau_i)$, we obtain that condition (4) is satisfied by $\mathcal{R}$.

This concludes the proof that $\mathsf{Aut}(\sigma)$ weakly timed simulates $\mathcal{A}$ which implies the language inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$.

($\supseteq$): Assume now that σ is a winning strategy in $\mathcal{G}_{\mathcal{A},(k,M')}$. Let us prove that $\mathcal{L}(\mathsf{Aut}(\sigma)) \subseteq \mathcal{L}(\mathcal{A})$.

Let $w \in \mathcal{L}(\mathsf{Aut}(\sigma))$. Then, there exists a run $\varrho'_w = S'_0 \xrightarrow{w}_{\mathsf{Aut}(\sigma)} S'_n$ in $\mathsf{Aut}(\sigma)$, such that S'_n is accepting. We want to prove that w also belongs to $\mathcal{L}(\mathcal{A})$. To do so, we first build a configuration path going from the initial configuration $(\ell_0, \bigwedge_{z,z' \in X \cup Y} z - z' = 0, \top, \{\overline{0}_Y\})$ to a configuration of S'_n whose location is accepting. This is performed backwards, using the definition the function $\mathsf{Succ}_e^{\varepsilon}$ for elementary successors. By Equation (3.4), this configuration sequence corresponds to a path in $\mathcal{A}$ (that is, an alternating sequence of locations and edges of $\mathcal{A}$). Then we show that along this path in $\mathcal{A}$, there exists a run of $\mathcal{A}$ reading w. This will allow us to conclude that $w \in \mathcal{L}(\mathcal{A})$ and thus $\mathcal{L}(\mathsf{Aut}(\sigma)) \subseteq \mathcal{L}(\mathcal{A})$.

Before the proof of these two steps, we introduce some notations. The accepting run over w in $\mathsf{Aut}(\sigma)$ is $\varrho'_w = S'_0 \xrightarrow{\tau_0}_{\mathsf{Aut}(\sigma)} \xrightarrow{a_1}_{\mathsf{Aut}(\sigma)} S'_1 \cdots S'_{n-1} \xrightarrow{\tau_{n-1}}_{\mathsf{Aut}(\sigma)} \xrightarrow{a_n}_{\mathsf{Aut}(\sigma)} S'_n$ with the following notation $w = (\sum_{l=0}^{j-1} \tau_l, a_j)_{1 \leq j \leq n}$. We further write $S'_i = (L_{S'_i}, v_{S'_i})$ for all $0 \leq i \leq n$ with $L_{S'_i} = (\mathcal{E}'_i, \mathsf{I}'_i)$ and denote by ℓ_γ, C_γ, b_γ and r_γ respectively the location, the relation, the boolean and the region of a configuration γ.

Construction of a path π in $\mathcal{A}$. Let j be a fixed index such that $1 \leq j \leq n$. Then, for every configuration in $\mathcal{E}'_j$, one can follow backwards the elementary successors by ε-transitions until a configuration which is the elementary successor of a configuration in $\mathcal{E}'_{j-1}$. Repeating this, one can backwards follow the whole run ϱ'_w. Formally, for every configuration $\gamma_j \in \mathcal{E}'_j$ marked $\top$ of S'_j there is a finite sequence $(\gamma_j^i)_{0 \leq i \leq n_j}$ of configurations marked $\top$ in $L_{S'_j}$ such that:

- there exists $\gamma_{j-1} \in S'_{j-1}$ such that $\gamma_j^0 \in \mathsf{Succ}_e^{\varepsilon}[r_{\gamma_j}^0, a_j, \sigma(L_{S'_{j-1}}, (r_{\gamma_j^0}, a_j))](\gamma_{j-1})$,

- $\gamma_j^{n_j} = \gamma_j$,
- for all $1 \leq i \leq n_j$, $\gamma_j^i \in \mathsf{Succ}_e^\varepsilon[r_{\gamma_j^i}, \varepsilon, \emptyset](\gamma_j^{i-1})$.

Remark that the fact that configurations are marked $\top$ is implied, by definition of $\mathsf{Succ}_e^\varepsilon$ (see Equation (3.4)) by the fact that γ_j itself is marked $\top$.

We can thus consider the configuration path π, corresponding to the entire run ϱ'_w starting the backward construction from an accepting configuration γ_n marked $\top$ in $L_{S'_n}$, because accepting locations of $\mathsf{Aut}(\sigma)$ are states containing at least one configuration whose location is accepting, and by definition of Bad because σ is winning. The path π is thus of the following form:

$\pi = (\ell_0, \bigwedge_{z,z' \in X \cup Y} z - z' = 0, \top, \{\overline{0}_Y\}) \xrightarrow{\varepsilon} \gamma_0^1 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \gamma_0^{n_0} \xrightarrow{a_1} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \gamma_{j-1}^0 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \gamma_{j-1}^{n_{j-1}} \xrightarrow{a_j} \gamma_j^0 \xrightarrow{\varepsilon} \cdots \xrightarrow{a_n} \gamma_n$.

Then, still by definition of the function $\mathsf{Succ}_e^\varepsilon$ in Equation (3.4), this configuration path corresponds to a path in $\mathcal{A}$. It is thus sufficient to prove that there is a run ϱ_π reading w in $\mathcal{A}$ along this path, that is:

$$\varrho_\pi = (\ell_0, \{\overline{0}\}) \xrightarrow{\tau_0^0} \xrightarrow{\varepsilon} (\ell_{\gamma_0^1}, v_0^1) \xrightarrow{\tau_0^1} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (\ell_{\gamma_0^{n_0}}, v_0^{n_0}) \xrightarrow{\tau_0^{n_0}} \xrightarrow{a_1} (\ell_{\gamma_1^0}, v_1^0) \xrightarrow{\tau_1^0} \xrightarrow{\varepsilon}$$
$$\cdots \xrightarrow{\varepsilon} (\ell_{\gamma_{j-1}^0}, v_{j-1}^0) \xrightarrow{\tau_{j-1}^0} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (\ell_{\gamma_{j-1}^{n_{j-1}}}, v_{j-1}^{n_{j-1}}) \xrightarrow{\tau_{j-1}^{n_{j-1}}} \xrightarrow{a_j} \cdots \xrightarrow{a_n} (\ell_{\gamma_n}, v_n)$$

where for all $0 \leq j \leq n-1$, $\sum_{i=1}^{n_j} = \tau_j$.

Reading w along π in $\mathcal{A}$. Let us prove that one can define delays along π to obtain a run in $\mathcal{A}$ reading w. We even prove a stronger fact: for each fragment of the path corresponding to one transition of the run ϱ'_w in $\mathsf{Aut}(\sigma)$, from any valuation $v \in \mathbb{R}_+^X$ in relation with the valuation $v_{S'_{j-1}} \in \mathbb{R}_+^Y$, one can define suitable delays. Formally, let us prove that for every $1 \leq j \leq n$, if $(v_{j-1}^0, v_{S'_{j-1}}) \in C_{\gamma_{j-1}^0}$ then there are delays (τ_{j-1}^i)'s such that $(\ell_{\gamma_{j-1}^0}, v_{j-1}^0) \xrightarrow{\tau_{j-1}^0} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (\ell_{\gamma_{j-1}^{n_{j-1}}}, v_{j-1}^{n_{j-1}}) \xrightarrow{\tau_{j-1}^{n_{j-1}}} \xrightarrow{a_j} (\ell_{\gamma_j^0}, v_j^0)$ is a run of $\mathcal{A}$ with $\sum_{i=1}^{n_j} \tau_{j-1}^i = \tau_j$ and $(v_j^0, \gamma_j^0) \in C_{\gamma_j^0}$. Observe that this property holds for $j = 1$ since $(\ell_0, \{\overline{0}_X\})$ corresponds to $(\ell_0, \overrightarrow{\{\overline{0}_{X \cup Y}\}}, \top, \{\overline{0}_Y\})$, formally $(\{\overline{0}_X\}, \{\overline{0_Y}\}) \in \overrightarrow{\{\overline{0}_{X \cup Y}\}}$.

The proof is structured as follows. We first show that invariants of $\mathcal{A}$ are satisfied in all the states corresponding to configurations of the path. Then we prove that transition a_j can be fired in states of $\mathcal{A}$ corresponding to the configuration $\gamma_{j-1}^{n_{j-1}}$ (with the associated valuation in ϱ'_w) and reach a state corresponding to γ_j^0. Finally we explain how to define delays in such a way that from any state corresponding to γ_{j-1}^0, one reaches a state corresponding to $\gamma_{j-1}^{n_{j-1}}$.

- **Invariants:** assuming that $(v_{j-1}^i + \tau_{j-1}^i, v_{S'_{j-1}} + \sum_{h=0}^i \tau_{j-1}^h) \in C_{\gamma_{j-1}^i}$, we prove that $v_{j-1}^i + \tau_{j-1}^i \in \mathsf{Inv}(\ell_{\gamma_{j-1}^i})$. Indeed, as γ_{j-1}^i is marked $\top$, $\left[[\mathsf{Inv}(\ell_{\gamma_{j-1}^i}) \cap C_{\gamma_{j-1}^i}]_{|Y} \cap C_{\gamma_{j-1}^i}\right]_{|X} = \mathsf{Inv}(\ell_{\gamma_{j-1}^i})$ and $[\mathsf{Inv}(\ell_{\gamma_{j-1}^i}) \cap C_{\gamma_{j-1}^i}]_{|Y} = \mathsf{I}'_{j-1}$ by definition of I'_{j-1} (Equation (3.6) on page 61). As a consequence $v_{S'_{j-1}} + \sum_{h=0}^i \tau_{j-1}^h \in \mathsf{I}'_{j-1}$ implies $v_{j-1}^i + \tau_{j-1}^i \in \mathsf{Inv}(\ell_{\gamma_{j-1}^i})$. In words, assuming that valuations in $\mathcal{A}$ satisfy corresponding relations with corresponding valuations in $\mathsf{Aut}(\sigma)$, invariants of locations of $\mathcal{A}$ are satisfied.

- **Discrete transitions labeled in Σ:** let $(\ell_{\gamma_{j-1}^{n_{j-1}}}, v_{j-1}^{n_{j-1}} + \tau_{j-1}^{n_{j-1}}) \xrightarrow{a_j} (\ell_{\gamma_j^0}, v_j^0)$ be a discrete transition of ϱ_π, labeled in Σ. Assuming that $(v_{j-1}^{n_{j-1}} + \tau_{j-1}^{n_{j-1}}, v_{S'_{j-1}} + \tau_{j-1}) \in C_{\gamma_{j-1}^{n_{j-1}}}$, we prove that it is a transition of $\mathcal{A}$ and that $(v_j^0, v_{S'_j}) \in C_{\gamma_j^0}$.

  By construction of ϱ_π, $\gamma_j^0 \in \mathsf{Succ}_e^\varepsilon[r_j, a_j, Y_j](\gamma_{j-1}^{n_{j-1}})$. Moreover, by definition of ϱ'_w, $v_{S'_{j-1}} + \tau_{j-1} \in r_j$. Then, by definition of $\mathsf{Succ}_e^\varepsilon$ in Section 3.3.2, and because γ_j^0 is marked $\top$, there exists an edge $(\ell_{\gamma_{j-1}^{n_{j-1}}}, g_j, a_j, X_j, \ell_{\gamma_j^0})$ in $\mathcal{A}$ such that conditions $\mathsf{D}_{\neq\emptyset}$ and $\mathsf{D}_\subseteq$ are satisfied, *i.e.* $\emptyset \neq [r_j \cap C_{\gamma_{j-1}^{n_{j-1}}}]_{|X} \subseteq g_j \cap \mathsf{Inv}(\ell_{\gamma_{j-1}^{n_{j-1}}}) \cap (\mathsf{Inv}(\ell_j^0))_{[X_j \leftarrow 0]^{-1}}$, and $C_{\gamma_j^0} = \mathsf{up}(r_j, C_{\gamma_{j-1}^{n_{j-1}}}, g_j, \ell_{\gamma_{j-1}^{n_{j-1}}}, \ell_{\gamma_j^0}, X_j, Y_j)$ (defined in Equation (3.8) on page 61). As a consequence, this edge is fireable from $(\ell_{\gamma_{j-1}^{n_{j-1}}}, v_{j-1}^{n_{j-1}} + \tau_{j-1}^{n_{j-1}})$, indeed $v_{j-1}^{n_{j-1}} + \tau_{j-1}^{n_{j-1}} \in [v_{S'_{j-1}} + \tau_{j-1} \cap C_{\gamma_{j-1}^{n_{j-1}}}]_{|X} \subseteq [r_j \cap C_{\gamma_{j-1}^{n_{j-1}}}]_{|X} \subseteq g_j \cap \mathsf{Inv}(\ell_{\gamma_{j-1}^{n_{j-1}}}) \cap (\mathsf{Inv}(\ell_j^0))_{[X_j \leftarrow 0]^{-1}}$. Finally, $(v_j^0, v_{S'_j}) \in C_{\gamma_j^0}$ by definition of up (Equation (3.8)).

- **Delays and ε-transitions:** let $(\ell_{\gamma_{j-1}^0}, v_{j-1}^0) \xrightarrow{\tau_{j-1}^0} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (\ell_{\gamma_{j-1}^{n_{j-1}}}, v_{j-1}^{n_{j-1}}) \xrightarrow{\tau_{j-1}^{n_{j-1}}} (\ell_{\gamma_{j-1}^{n_{j-1}}}, v_{j-1}^{n_{j-1}} + \tau_{j-1}^{n_{j-1}})$ be a sequence of delays and ε-transitions of ϱ_π corresponding to the delay τ_{j-1} of ϱ'_w in $\mathsf{Aut}(\sigma)$. Assuming that $(v_{j-1}^0, v_{S'_{j-1}}) \in C_{\gamma_{j-1}^0}$, we prove that one can fix τ_{j-1}^i's such that this is a sequence of transitions of $\mathcal{A}$, $\sum_{h=0}^{n_{j-1}-1} \tau_{j-1}^h = \tau_{j-1}$ and $(v_{j-1}^{n_{j-1}} + \tau_{j-1}^{n_{j-1}}, v_{S'_{j-1}} + \tau_{j-1}) \in C_{\gamma_{j-1}^{n_{j-1}}}$.

  Note that $v_{S'_{j-1}} \in r_{\gamma_{j-1}^{n_{j-1}}}$ and $v_{S'_{j-1}} + \tau_j \in r_j \subseteq \overrightarrow{r_{\gamma_{j-1}^{n_{j-1}}}}$. Let us define τ_{j-1}^i as follows: $\tau_{j-1}^{n_{j-1}} = \tau_j - \sum_{i=0}^{n_{j-1}-1} \tau_{j-1}^i$ and for all $0 \leq i < n_{j-1}$, $\tau_{j-1}^i = 0$ if $v_{S'_{j-1}} + \sum_{h=0}^{i-1} \tau_{j-1}^h \in r_{j-1}^i$, otherwise we fix τ_{j-1}^i as any delay such that $v_{S'_{j-1}} + \sum_{h=0}^{i} \tau_{j-1}^h \in r_{j-1}^i$ and $v_{S'_{j-1}} + \sum_{h=0}^{i} \tau_{j-1}^h \leq \tau_{j-1}$.

  Let us prove by induction over i that for all $0 \leq i < n_{j-1}$, $(\ell_{\gamma_{j-1}^i}, v_{j-1}^i) \xrightarrow{\tau_{j-1}^i} \xrightarrow{\varepsilon} (\ell_{\gamma_{j-1}^{i+1}}, v_{j-1}^{i+1})$ is a transition of $\mathcal{A}$ and $(v_{j-1}^{i+1}, v_{S'_{j-1}} + \sum_{h=0}^{i} \tau_{j-1}^h) \in C_{\gamma_{j-1}^{i+1}}$.

  First of all, we initialize thanks to the assumption $(v_{j-1}^0, v_{S'_{j-1}}) \in C_{\gamma_{j-1}^0}$.

  Let us fix $0 \leq i < n_{j-1}$ and assume that $(v_{j-1}^i + \tau_{j-1}^i, v_{S'_{j-1}} + \sum_{h=0}^{i} \tau_{j-1}^h) \in C_{\gamma_{j-1}^i}$. Hence $(v_{j-1}^i + \tau_{j-1}^i, v_{S'_{j-1}} + \sum_{h=0}^{i+1} \tau_{j-1}^h) \in C_{\gamma_{j-1}^i}$. Then, we can conclude about the inductive step in the same way as in the previous step for a_j's.

  We obtain that it is a sequence of transitions of $\mathcal{A}$ and that $(v_{j-1}^{n_{j-1}}, v_{S'_{j-1}} + \sum_{h=0}^{n_{j-1}-1} \tau_{j-1}^h) \in C_{\gamma_{j-1}^{n_{j-1}}}$, which implies that $(v_{j-1}^{n_{j-1}} + \tau_{j-1}^{n_{j-1}}, v_{S'_{j-1}} + \tau_{j-1}) \in C_{\gamma_{j-1}^{n_{j-1}}}$. □

3.3.4 Comparison with [KT09]

In Section 3.2, we compared our approach with existing methods in the restricted case where timed automata neither have invariants nor ε-transitions. The determinization procedure of [BBBB09] does not deal with invariants and ε-transitions. We therefore compare our extended approach only with the over-approximation algorithm of [KT09].

The models in [KT09] are timed automata with silent transitions, and actions are classified with respect to their urgency: eager, lazy or delayable. First of all, the authors propose an ε-closure computation which does not terminate in general, and rely on the fact that termination can be ensured by some abstraction. Second, the urgency in the model is not preserved by the over-approximation construction: the resulting deterministic timed automaton only contains lazy transitions (intuitively lazy over-approximates all kinds of urgency). Note that we classically decided to rather use invariants to model urgency, but our approach could be adapted to the same model as in [KT09], and would preserve urgency more often, the same way as we do for invariants. These observations underline the benefits of our game-based approach for timed automata with invariants and ε-transitions compared to existing work.

3.4 Beyond over-approximation

In the previous sections, we presented a game-based approach which yields a deterministic over-approximation of a given timed automaton. Yet, we advocate that over-approximations are not always appropriate, and, depending on the context, under-approximations or other approximations might be more suitable. We therefore explain in this section how to adapt our framework in order to generate deterministic under-approximations, and also combine over- and under-approximations.

3.4.1 Under-approximation

One motivation for building deterministic under-approximations of a regular timed language is that one can decide whether the timed language is approximated provided that the 'largest' language is recognized by a deterministic timed automaton. Therefore, given $\mathcal{A}$ a non-deterministic timed automaton, for every deterministic under-approximation $\mathcal{B}$, one can decide whether the approximation is strict or not, that is whether the reverse inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ also holds. Contrary to what happens for over-approximations, one would thus be able to detect if a losing strategy yet yields a deterministic equivalent to the original timed automaton.

We now briefly explain how to modify the game construction, so that any strategy yields an under-approximation, and any winning strategy provides a deterministic equivalent. When we aim at building an over-approximation, during the construction of the game, all litigious successors (i.e. configurations marked $\bot$) are built, possibly introducing more behaviors than in the original TA. In order to obtain an under-approximation, the litigious successors are simply not constructed. Also, ε-transitions and invariants are not more difficult to handle: (1) the ε-closure is under-approximated (by avoiding to built configurations marked $\bot$); (2) the invariant of a state is redefined as the union of all regions such that the induced guard is included in the invariant of the location of some configuration marked $\top$; and (3) finally, the set Bad is defined as the set of states where either some litigious successor existed (but was not built), or for which the invariant or the ε-closure has been under-approximated. We do not give the complete details of this construction, since in the next subsection we present an extension that subsumes both over-approximations and under-approximations.

3.4.2 Combining over- and under-approximation

Beyond over-approximations and under-approximations, combinations of both can be meaningful in some contexts. Model-based testing is an example of such contexts. Given a non-deterministic timed automaton $\mathcal{A}$, it can be proved that a deterministic timed automaton $\mathcal{B}$ which over-approximates outputs and under-approximates inputs, preserves the conformance relation tioco. As a consequence, test

cases can be generated from $\mathcal{B}$, as sound test cases for $\mathcal{B}$ remain sound for $\mathcal{A}$. Details on the application of deterministic approximations to the test generation from non-deterministic timed automata models can be found in Chapter 4.

Let us now explain more generally how to combine over-approximations and under-approximations. To this aim, we consider timed automata with a partitioned alphabet, $\Sigma = \Sigma_1 \sqcup \Sigma_2$, and introduce the notion of (Σ_1, Σ_2)-refinement relation. Note that (Σ_1, Σ_2)-refinement is defined here only for a pair of timed automata $(\mathcal{A}, \mathcal{A}')$ when $\mathcal{A}'$ has no ε-transition.

First of all, let us introduce some notations to shorten the definition. For any timed word w, we write $s_0 \overset{w}{\Rightarrow} s$ if there exists a run from s_0 to s reading w. If s is left implicit, we write $s_0 \overset{w}{\Rightarrow}$, thus meaning that there exists a state s such that $s_0 \overset{w}{\Rightarrow} s$. We also use this convention for the transition relation $\rightarrow$.

Definition 3.5. *Let $\mathcal{A}$ be a timed automaton and $\mathcal{A}'$ be a timed automaton without ε-transitions over the alphabet $\Sigma = \Sigma_1 \sqcup \Sigma_2$, and $\mathcal{T}_{\mathcal{A}} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \rightarrow_{\mathcal{A}})$, $\mathcal{T}_{\mathcal{A}'} = (S', s'_0, S'_F, (\mathbb{R}_+ \times \Sigma), \rightarrow_{\mathcal{A}'})$ their associated transition systems. We say that $\mathcal{A}$ (Σ_1, Σ_2)-refines $\mathcal{A}'$ and write $\mathcal{A} \preceq \mathcal{A}'$ if:*

1. *if $s_0 \overset{w}{\Rightarrow}_{\mathcal{A}} \xrightarrow{\tau_0}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \xrightarrow{\tau_n}_{\mathcal{A}}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$, and $s'_0 \overset{w}{\Rightarrow}_{\mathcal{A}'}$ then $s'_0 \overset{w}{\Rightarrow}_{\mathcal{A}'} \xrightarrow{\tau}_{\mathcal{A}'}$ with $\tau = \sum_{i=0}^{n} \tau_i$;*

2. *if $s_0 \xRightarrow{w.(t,a_2)}_{\mathcal{A}}$ where $a_2 \in \Sigma_2$, and $s'_0 \overset{w}{\Rightarrow}_{\mathcal{A}'}$ then $s'_0 \xRightarrow{w.(t,a_2)}_{\mathcal{A}'}$;*

3. *if $s'_0 \xRightarrow{w.(t,a_1)}_{\mathcal{A}'}$ where $a_1 \in \Sigma_1$, and $s_0 \overset{w}{\Rightarrow}_{\mathcal{A}} \xrightarrow{\tau_0}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \xrightarrow{\tau_n}_{\mathcal{A}}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$ and the accumulated delay of w is $t - \sum_{i=0}^{n} \tau_i$, then $s_0 \xRightarrow{w.(t,a_1)}_{\mathcal{A}}$.*

Intuitively, the (Σ_1, Σ_2)-refinement is in the spirit of the alternating simulation from [AHKV98], with timing aspects. Apart from time, the notable difference is that (Σ_1, Σ_2)-refinement is defined at a language level and is not a binary relation between states. Roughly speaking, the link between refinement and alternating simulation is the same as between language inclusion and simulation.

Let us explain the three properties of the definition. The first property specifies that if a given word can be read in both timed automata and that a delay τ can be observed in $\mathcal{A}$ (through possible ε-transitions, then such a delay τ can also be observed in $\mathcal{A}'$. No ε-transitions are allowed in $\mathcal{A}'$ for readability, but a natural extension of this definition can be easily written allowing them. The second property states that if a given word can be read in $\mathcal{A}$ and $\mathcal{A}'$ can read this word except up to the last action, and if this last action belongs to Σ_2, then $\mathcal{A}'$ has to be able to read the complete word. These two properties thus express that $\mathcal{A}'$ simulates $\mathcal{A}$, on a language level for Σ_2-actions and delays. Last, the third requirement states the simulation of $\mathcal{A}'$ by $\mathcal{A}$ on a language level for Σ_1-actions.

Remark that even if there is no ε-transitions in $\mathcal{A}$, the definition is not symmetric: $\mathcal{A}$ (Σ_1, Σ_2)-refines $\mathcal{A}'$ does not imply that $\mathcal{A}'$ (Σ_2, Σ_1)-refines $\mathcal{A}$, due to the way delays are taken care of. Our targeted application to test selection is responsible for this choice. In particular, if Σ_1 and Σ_2 consist respectively of the input and output alphabets, the (Σ_1, Σ_2)-refinement relation generalizes the io-refinement relation between deterministic timed automata introduced in [DLL$^+$10], which was inspired by the alternating simulation [AHKV98]. Then, the inverse relation (which we refer to as generalized io-abstraction) still preserves the **tioco** conformance relation [KT09]: implementations that conform to a specification also conform to any io-abstraction of this specification. As a consequence soundness of test cases is preserved by io-refinement: a test suite which is sound for a given specification is also sound for any io-refinement of the specification.

Our goal here is to combine over- and under-approximations in the construction of the game so that any strategy for Determinizator yields a deterministic (Σ_1, Σ_2)-abstraction of the original automaton. The game construction is adapted: transitions over actions of Σ_2 and invariants are over-approximated, whereas transitions over actions of Σ_1 are under-approximated. The definition of the (Σ_1, Σ_2)-refinement imposes global, *i.e.* language-based, conditions. As a consequence, when performing an over-approximation, information about configurations which are removed by an under-approximation must be kept. Moreover, to deal with ε-transitions, the ε-closure should be over-approximated before a Σ_2-action, and under-approximated before a Σ_1-action. As a consequence, the structure of the states of Spoiler is enriched as follows. The set of configurations is replaced by a pair of sets which are respectively the over-approximation of the set of configurations and the set of configurations built after successive over- and under-approximations, depending on the moves leading to its construction to which we applied the under-approximating ε-closure. The invariant associated with a state of Spoiler is then defined in the same way as before, using the first set of configurations (the over-approximation). Formally, given a state of Spoiler whose first set of configurations is $\{(\ell_j, C_j, b_j, r_j)\}_j$, the invariant is defined as follows:

$$\mathsf{I} = \bigcup \{ r'' \in \mathsf{Reg}^Y_{M'} \mid \exists j,\ r'' \in \overrightarrow{r_j} \wedge [r'' \cap C_j]_{|X} \cap \mathsf{Inv}(\ell_j) \neq \emptyset \}. \tag{3.9}$$

The over-approximation of the invariants is compatible with under-approximations of some behaviors, since guards always are intersected with the original invariants, rather than the approximated one, in the construction of the game. However, under-approximating invariants could hinder over-approximations by constraining too much the guards.

Before giving the formal definition of the game, we introduce the two elementary successor operators (one for over-approximation, the other for under-approximation) as well as the two ε-closure operators. Given (ℓ, C, b, r) a configuration such that r' is a time-successor of r, we detail the computation of elementary successors depending on a. If $a \in \Sigma_2$, its elementary successors set by (r', a) and Y' is:

$$\mathsf{Succ}^+_e[r', a, Y'](\ell, C, b, r) = \left\{ (\ell', C', b', r'_{[Y' \leftarrow 0]}) \mid \begin{array}{l} \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ such that} \\ [r' \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}} \neq \emptyset \\ C' = \mathsf{up}(r', C, g, \mathsf{Inv}(\ell), \mathsf{Inv}(\ell'), X', Y') \\ b' = b \wedge ([r' \cap C]_{|X} \subseteq g) \end{array} \right\} \tag{3.10}$$

Now, if $a \in \Sigma_1$, its elementary successors set by (r', a) and Y' is:

$$\mathsf{Succ}^-_e[r', a, Y'](\ell, C, b, r) = \left\{ (\ell', C', b', r'_{[Y' \leftarrow 0]}) \mid \begin{array}{l} \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ such that} \\ [r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}} \\ C' = \mathsf{up}(r', C, g, \mathsf{Inv}(\ell), \mathsf{Inv}(\ell'), X', Y') \\ b' = b \end{array} \right\} \tag{3.11}$$

In both definitions, $\mathsf{up}(r', C, g, \mathsf{Inv}(\ell), \mathsf{Inv}(\ell'), X', Y')$ is the update of the relation C between clocks in X and Y after the moves of the two players, that is after taking action a in r', resetting $X' \subseteq X$ and $Y' \subseteq Y$, and forcing the satisfaction of g, $\mathsf{Inv}(\ell)$ and $\mathsf{Inv}(\ell')$. The formal definition is given in Equation (3.8) page 61.

Roughly, Succ_e^+ yields a set of configurations over-approximating the set of successor states in $\mathcal{A}$. Indeed, successor configurations are built as soon as $\mathsf{D}_{\neq\emptyset}$ is satisfied. On the other side, Succ_e^- under-approximates the set of states using the restrictive condition $\mathsf{D}_{\subseteq}$.

To formalize over- and under-approximated ε-closures of a set of configurations, we define ε-closures of a single configuration. The closure of a set of configurations being the union of the closures of the individual configurations. Given (ℓ, C, b, r) a configuration, its ε-closures noted $\mathsf{cl}_\varepsilon^+(\ell, C, b, r)$ and $\mathsf{cl}_\varepsilon^-(\ell, C, b, r)$, are the smallest fixpoints of the functionals

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{\substack{(\ell', C', b', r') \in X \\ r'' \in \overrightarrow{r'}}} \mathsf{Succ}_e^+[r'', \varepsilon, \emptyset](\ell', C', b', r'), \text{ and} \tag{3.12}$$

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{\substack{(\ell', C', b', r') \in X \\ r'' \in \overrightarrow{r'}}} \mathsf{Succ}_e^-[r'', \varepsilon, \emptyset](\ell', C', b', r'), \text{respectively.} \tag{3.13}$$

We are now in the position to provide the formal definition of the game.

Definition 3.6. *Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E, \mathsf{Inv})$ be a timed automaton and (k, M') resources. We let $\mathbf{M} = \max(M, M')$, and Y a set of k clocks. The game associated with $\mathcal{A}$ and (k, M') is $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ where:*

- V *is a finite set of vertices, partitioned into V_S (vertices of Spoiler) and V_D (vertices of Determinizator). $\mathsf{V}_S \subseteq (2^{L \times \mathsf{Rel}_{\mathbf{M}}(X \cup Y) \times \{\top,\bot\} \times \mathsf{Reg}_{M'}^Y})^2 \times I_{M'}(Y)$ and $\mathsf{V}_D \subseteq \mathsf{V}_S \times \mathsf{Reg}_{M'}^Y \times \Sigma$,*
- $\mathsf{v}_0 = ((\mathsf{cl}_\varepsilon^+, \mathsf{cl}_\varepsilon^-)(\{(\ell_0, \bigwedge_{z,z' \in X \cup Y} z - z' = 0, \top, \{\overline{0}\})\}), \mathsf{I}_0)$ *with I_0 the invariant from Equation* (3.9)*, is the initial vertex and belongs to player Spoiler;*
- Act *is the set of possible actions partitioned into $\mathsf{Act}_S = \mathsf{Reg}_{M'}^Y \times \Sigma$ and $\mathsf{Act}_D = 2^Y$;*
- *$\delta = \delta_S \cup \delta_D$ is the transition relation with δ_S and δ_D defined as follows.*
  - *$\delta_S \subseteq \mathsf{V}_S \times \mathsf{Act}_S \times \mathsf{V}_D$ is the set of edges of the form $\mathsf{v}_S \xrightarrow{(r',a)} (\mathsf{v}_S, (r', a))$ for $\mathsf{v}_S = ((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I})$ and*
    - *if $a \in \Sigma_1$ and $\exists(\ell, C, \top, r) \in \mathcal{E}^2$ such that $r' \in \overrightarrow{r}$ and one of the two following conditions is satisfied*
      - *$\exists \ell \xrightarrow{g,a,X'} \ell' \in E$ such that $\mathsf{D}_{\subseteq}$ is satisfied,* i.e. $[r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}}$,
      - *$\forall(\ell, C, b, r) \in \mathcal{E}^1$, $\exists \ell \xrightarrow{g,a,X'} \ell' \in E$ such that the condition $\mathsf{D}_{\subseteq}$ is satisfied,* i.e. $[r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}}$*; or*
    - *if $a \in \Sigma_2$ and $\exists(\ell, C, b, r) \in \mathcal{E}^1$ such that $r' \in \overrightarrow{r}$ and $\mathsf{D}_{\neq\emptyset}$ is satisfied,* i.e. $\exists \ell \xrightarrow{g,a,X'} \ell' \in E$ *s.t.* $[r' \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}} \neq \emptyset$*;*
  - *$\delta_D \subseteq \mathsf{V}_D \times \mathsf{Act}_D \times \mathsf{V}_S$ is the set of edges of the form $\mathsf{v}_D \xrightarrow{(Y')} ((\mathcal{E}'^1, \mathcal{E}'^2), \mathsf{I}')$ for $\mathsf{v}_D = (((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}), (r', a_1))$ and*

  - *if $a \in \Sigma_1$ and the target state satisfies the following conditions:*
    $\mathcal{E}'^1 = \mathsf{cl}^+_\varepsilon(\cup_{\gamma\in\mathcal{E}^1}\mathsf{Succ}^+_e[r',a,Y'](\gamma))$, $\mathcal{E}'^2 = \mathsf{cl}^-_\varepsilon(\cup_{\gamma\in\mathcal{E}^2}\mathsf{Succ}^-_e[r',a,Y'](\gamma))$ and I' is defined above in Equation (3.9)*; or*
  - *if $a \in \Sigma_2$ and the target state satisfies the following conditions:*
    $\mathcal{E}'^1 = \mathsf{cl}^+_\varepsilon(\cup_{\gamma\in\mathcal{E}^1}\mathsf{Succ}^+_e[r',a,Y'](\gamma))$, $\mathcal{E}'^2 = \mathsf{cl}^-_\varepsilon(\cup_{\gamma\in\mathcal{E}^1}\mathsf{Succ}^+_e[r',a,Y'](\gamma))$ and I' is defined above in Equation (3.9)*;*

- $\mathsf{Bad} = \{(\{(\ell_j, C_j, \bot, r_j)\}_j, \mathcal{E}^2), \mathsf{I})\}$
  $\cup \{(\{(\ell_j, C_j, b_j, r_j)\}_j, \mathcal{E}^2), \mathsf{I}) \mid \forall h \ \big((\forall j,\ r_j \in \overrightarrow{r_h}) \Rightarrow (\ell_h \in F \Rightarrow b_h = \bot)\big)$
  $\wedge\ (\exists i,\ \ell_i \in F)\}$
  $\cup \{((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}) \mid \exists s \in \mathcal{E}^1,\ a \in \Sigma_1,\ r' \text{ and } Y' \text{ s.t. } \mathsf{Succ}^+_e[r',a,Y'](s) \neq \emptyset$
  $\wedge\ ((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}) \xrightarrow{(r',a)}\}$

  is the set of bad states.

In words, the possible moves of the players are defined as follows. Given $\mathsf{v}_S = ((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}) \in \mathsf{V}_S$ a state of Spoiler and (r', a) one of his moves, the successor state is defined as a state $\mathsf{v}_D = (\mathsf{v}_S, (r', a)) \in \mathsf{V}_D$. Note that v_D is built only if a condition depending on a is satisfied:

- if $a \in \Sigma_1$, then one wants to under-approximate the behaviors. To force the under-approximation, v_D is defined only if, either there is a configuration marked $\top$ in $\mathcal{E}^2$ from which a can be fired without approximation, or from all the configurations in $\mathcal{E}^1$, a can be fired.

- when $a \in \Sigma_2$, the goal is to over-approximate, thus v_D is built if there is at least one configuration in $\mathcal{E}^1$ from which a can be fired.

Given $\mathsf{v}_D = (\mathsf{v}_S, (r', a)) \in \mathsf{V}_D$ a state of Determinizator and $Y' \subseteq Y$ one of its moves, the successor state is $\mathsf{v}'_S = ((\mathcal{E}'^1, \mathcal{E}'^2), \mathsf{I}') \in \mathsf{V}_S$ such that $\mathcal{E}'^1$ is the over-approximation of successor configurations of $\mathcal{E}^1$, and $\mathcal{E}'^2$ is the set of successor configurations obtained by successive over-approximations and under-approximations (depending on the actions). In particular, the ε-closure of $\mathcal{E}'^1$ is over-approximated whereas the ε-closure of $\mathcal{E}'^2$ is under-approximated.

Last, in order to preserve exactness of winning strategies for Determinizator, the set Bad is extended: states obtained before an under-approximation, that is from which some behaviors are cut, are added to Bad. More precisely, any state of Spoiler v_S containing a configuration (ℓ, C, b, r) in the over-approximating set of configurations such that, for (r', a_1) and Y' moves of the two players $\mathsf{Succ}^+_e[r', a_1, Y'](\ell, C, b, r)$ is not empty whereas the successor is not built, is in Bad.

Under all these modifications of the game, the following proposition holds:

Proposition 3.4. *Let $\mathcal{A}$ be a timed automaton over the alphabet $\Sigma = \Sigma_1 \sqcup \Sigma_2$, and (k, M') resources. For every strategy σ of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$, $\mathsf{Aut}(\sigma)$ is a deterministic timed automaton over resources (k, M') and satisfies $\mathcal{A} \preceq \mathsf{Aut}(\sigma)$. Moreover, if σ is winning, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$.*

Proof. The difficult part of the proof concerns arbitrary strategies. Assuming σ is a strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$, let us prove that $\mathsf{Aut}(\sigma)$ is a (Σ_1, Σ_2)-abstraction of $\mathcal{A}$, that is, $\mathcal{A} \preceq \mathsf{Aut}(\sigma)$. Let $\mathcal{T}_\mathcal{A} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \to_\mathcal{A})$, $\mathcal{T}_{\mathsf{Aut}(\sigma)} = (\mathsf{S}', S'_0, \mathsf{S}'_F, (\mathbb{R}_+ \times \Sigma), \to_{\mathsf{Aut}(\sigma)})$, the respective timed transition systems associated with $\mathcal{A}$ and $\mathsf{Aut}(\sigma)$.

Recall the three properties that we have to prove:

1. if $s_0 \overset{w}{\Rightarrow}_\mathcal{A} \xrightarrow{\tau_0}_\mathcal{A} \xrightarrow{\varepsilon}_\mathcal{A} \cdots \xrightarrow{\tau_{n-1}}_\mathcal{A} \xrightarrow{\varepsilon}_\mathcal{A} \xrightarrow{\tau_n}_\mathcal{A}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$ and $S'_0 \overset{w}{\Rightarrow}_{\mathsf{Aut}(\sigma)}$, then $S'_0 \overset{w}{\Rightarrow}_{\mathsf{Aut}(\sigma)} \xrightarrow{\tau}_{\mathsf{Aut}(\sigma)}$ with $\tau = \sum_{i=0}^n \tau_i$;

2. if $s_0 \xRightarrow{w.(t,a_2)}_{\mathcal{A}}$ where $a_2 \in \Sigma_2$, and $S'_0 \xRightarrow{w}_{\mathsf{Aut}(\sigma)}$ then $S'_0 \xRightarrow{w.(t,a_2)}_{\mathsf{Aut}(\sigma)}$;

3. if $S'_0 \xRightarrow{w.(t,a_1)}_{\mathsf{Aut}(\sigma)}$ where $a_1 \in \Sigma_1$, and $s_0 \xRightarrow{w}_{\mathcal{A}} \xrightarrow{\tau_0}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \xrightarrow{\tau_n}_{\mathcal{A}}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$ and the accumulated delay of w is $t - \sum_{i=0}^{n} \tau_i$, then $s_0 \xRightarrow{w.(t,a_1)}_{\mathcal{A}}$.

Note that the proof of Theorem 3.2 on page 63 applies to prove the first and the second properties. We define the same binary relation $\mathcal{R} \subseteq S \times \mathsf{S}'$:

$$\mathcal{R} = \big\{ \big((\ell, v), ((\mathcal{E}^1, \mathcal{E}^2, \mathsf{l}), v')\big) \mid \exists (\ell, C, b, r) \in \mathcal{E}^1,\ (v, v') \in C \wedge v' \in \overrightarrow{r} \big\}.$$

By induction, $s_0 \xRightarrow{w}_{\mathcal{A}} s$ and $S'_0 \xRightarrow{w}_{\mathsf{Aut}(\sigma)} S'$ imply $(s, ((\mathcal{E}^1_{S'}, \mathsf{l}_{S'}), v_{S'})) \in \mathcal{R}$ assuming $S' = (((\mathcal{E}^1_{S'}, \mathcal{E}^2_{S'}), \mathsf{l}_{S'}), v_{S'})$. In words, the state estimate $\mathcal{E}^1_{S'}$ is exactly the contents of the state one would have in the game for over-approximations only.

The heart of the proof thus concerns the third property. In fact, we prove a stronger property:

3′. if $S'_0 \xRightarrow{w.(t,a_1)}_{\mathsf{Aut}(\sigma)}$ where $a_1 \in \Sigma_1$, and $s_0 \xRightarrow{w}_{\mathcal{A}}$ then $s_0 \xRightarrow{w.(t,a_1)}_{\mathcal{A}}$.

Indeed, we prove that if a timed word ending with an action in Σ_1 can be read in $\mathcal{A}'$ and its largest strict prefix can be read in $\mathcal{A}$, then the entire word can be read in $\mathcal{A}$. Let us assume that $S'_0 \xRightarrow{w\cdot(t,a_1)}_{\mathsf{Aut}(\sigma)}$ with $a_1 \in \Sigma_1$, and that $s_0 \xRightarrow{w}_{\mathcal{A}}$, and write S' for the state of $\mathsf{Aut}(\sigma)$ such that $S'_0 \xRightarrow{w}_{\mathsf{Aut}(\sigma)} S'$. Note that S' is unique because $\mathsf{Aut}(\sigma)$ is deterministic. Writing $S' = (((\mathcal{E}^1_{S'}, \mathcal{E}^2_{S'}), \mathsf{l}_{S'}), v_{S'})$, the a_1-transition from S' corresponds to an edge $(((\mathcal{E}^1_{S'}, \mathcal{E}^2_{S'}), \mathsf{l}_{S'}), (r', a_1, Y'), \mathsf{v})$ of $\mathsf{Aut}(\sigma)$. Hence there are two cases. Either (i) there exists $(\ell, C, \top, r) \in \mathcal{E}^2_{S'}$ such that $r' \in \overrightarrow{r}$ and there exists an edge $\ell \xrightarrow{g,a,X'} \ell'$ in $\mathcal{A}$ such that condition $\mathsf{D}_{\subseteq}$ is satisfied, that is $[r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}}$; Or (ii) for all $(\ell, C, b, r) \in \mathcal{E}^1_{S'}$ such that $r' \in \overrightarrow{r}$, there exists an edge $\ell \xrightarrow{g,a,X'} \ell'$ in $\mathcal{A}$ such that condition $\mathsf{D}_{\subseteq}$ holds.

- (i) In this case, the second part of the proof of Theorem 3.2 applies. One can thus build a path in $\mathcal{A}$ along which it is possible to read w, ending in a state of the form $(\ell, \tilde{v})$, such that $(\tilde{v}, v_{S'}) \in C$. As a consequence, $(\tilde{v} + \tau, v_{S'} + \tau) \in C$, where τ is the delay right before a_1 in $w(t, a_1)$, and thus $v_{S'} + \tau \in r'$. Since there exists $\ell \xrightarrow{g,a,X'} \ell'$ in $\mathcal{A}$ such that $\mathsf{D}_{\subseteq}$, $(\ell, \tilde{v} + \tau) \xrightarrow{a_1}_{\mathcal{A}} (\ell', \tilde{v} + \tau_{|X'})$ and in particular invariants of ℓ and ℓ' are satisfied. Hence $s_0 \xRightarrow{w\cdot(t,a_1)}_{\mathcal{A}} (\ell', \tilde{v} + \tau_{[X' \leftarrow 0]})$.
- (ii) By assumption, $S'_0 \xRightarrow{w}_{\mathsf{Aut}(\sigma)} S'$ and $s_0 \xRightarrow{w}_{\mathcal{A}} s$, then, as explained above, $(s, ((\mathcal{E}^1_{S'}, \mathsf{l}_{S'}), v'_S)) \in \mathcal{R}$. Then, $\mathcal{E}^1_{S'}$ contains a configuration of the form (ℓ, C, b, r) with $s = (\ell, v_s)$, and by (ii), there exists an edge $\ell \xrightarrow{g,a,X'} \ell' \in E$ such that $\mathsf{D}_{\subseteq}$ is satisfied. As a consequence, by the same reasoning as above, $s_0 \xRightarrow{w\cdot(t,a_1)}_{\mathcal{A}} (\ell', v_s + \tau_{[X' \leftarrow 0]})$.

We thus proved that $\mathcal{A} \preceq \mathsf{Aut}(\sigma)$.

Assume now that σ is winning. Thanks to the new definition of the set Bad, in this case we recover the properties of the original method. Indeed, by definition of Bad, for all locations $((\mathcal{E}^1, \mathcal{E}^2), \mathsf{l})$ of $\mathsf{Aut}(\sigma)$, on the one hand, $(\mathcal{E}^1, \mathsf{l})$ is a state of the game built with only over-approximations (see Section 3.3.2), which is not a bad state, and, on the other hand, this state has the same successor as in the original game because of the inclusion

$$\big\{((\mathcal{E}^1, \mathcal{E}^2), \mathsf{l}) \mid \exists s \in \mathcal{E}^1,\ \exists a \in \Sigma_1,\ \exists r',\ \exists Y' \text{ s.t. } \mathsf{Succ}^+_e[r', a, Y'](s) \neq \emptyset \wedge ((\mathcal{E}^1, \mathcal{E}^2), \mathsf{l}) \xrightarrow{(r',a)} \big\} \subseteq \mathsf{Bad}.$$

Therefore the proof of Theorem 3.2 applies and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$. □

3.5 Implementation of a prototype tool

We implemented a prototype tool during a visit in the team of Kim G. Larsen at Aalborg University, and in particular thanks to the help of Peter Bulychev. Currently, the implementations still has some limitations, it does not deal with invariants and ε-transitions and it only searches for a winning strategy. It allows to make some experiments over small examples, and in particular it has been used to compute all deterministic timed automata illustrating our approach.

3.5.1 Using zones instead of regions

In the approach described in this thesis, in order to maximize the chances to exactly determinize a timed automaton, the constructed deterministic timed automaton is split in regions. Unfortunately, the number of regions is exponential in the number of clocks. Moreover, there may be a lot regions which could be treated in the same way. For example, if the initial non-deterministic timed automaton has only constants 1 and 100 in its guards, then it is very costly to use regions, whereas it does not necessarily help for the determinization. More generally, if there are several clocks or large constants, regions imply a real explosion of the number of transitions leaving a location, which is often useless.

To avoid to use regions which have an intractable nature and to obtain a more reasonable number of transitions, we implemented an extension of the game approach to unions of regions, called zones. The counterpart is that, in some cases, determinization is less precise. The extension to zones is simple and the proof of the properties of the game can be done in the same way as for regions. The single constraint over zones is to form a partition of the set of valuations over Y. In fact, if two distinct zones intersect, the resulting timed automaton can be non-deterministic and if the set of zones does not cover the set of valuations, some words can be forgotten. The first intuition could be that using zones instead of regions decreases the number of possible moves for Spoiler. In fact, it rather forces Determinizator to have the same answer to the different moves of Spoiler, by merging them in a single move.

In our prototype, zones are induced by a set P of predicates over Y given as parameter with the non-deterministic timed automaton, the set of clocks Y and the maximal constant M for the output deterministic timed automaton. Each zone used as a move of Spoiler is the set of valuations satisfying a conjunction of predicates or their negations. For the determinization of some timed automaton, we cannot guess the optimized resources in general. Similarly, the selection of the best set of zones is a real problem even if the observation of the initial non-deterministic timed automaton can help. In the case where resources (Y, N, P) are not sufficient, that is there is no winning strategy for determinizator, the prototype yields deterministic under- and over-approximations constructed in the same way as in the region-based algorithm.

3.5.2 Implementation of the prototype

Our prototype tool is implemented in Python. Zones are coded by Difference Bounded Matrices (DBMs).

Definition 3.7 (Difference Bounded Matrix)**.** *A* Difference Bounded Matrix *over the set of n clocks X is an $(n+1)$-square matrix of pairs $(m, \precsim)$ with $\precsim \in \{<, \leq\}$ and $m \in \mathbb{Z} \cup \{+\infty\}$.*

The semantics of a DBM $\mathcal{M} = (m_{i,j}, \prec_{i,j})_{0 \leq i,j \leq n}$ over a set of clocks $\{x_1, x_2, \cdots, x_n\}$, is the zone defined by the constraint $\wedge_{0 \leq i,j \leq n} x_i - x_j \prec_{i,j} m_{i,j}$ with the convention $x_0 = 0$.

Usual operations over DBMs are efficiently performed thanks to the Python binding for the UPPAAL DBM library [UDL].

Given a non-deterministic timed automaton and resources for the determinization, the game may be huge. As a consequence, we try to avoid to construct all the states in the game. To do so, we implemented the on-the-fly algorithm proposed in [LS98] for safety games.

The program is quite short, it contains 500 lines of code. It consists of the following classes:

- Configuration
- StateOfSpoiler
- StateOfDeterminizator
- GameOfDeterminization

Class Configuration Attributes of a Configuration are a location, a relation and a boolean marking. Given a zone associated with this Configuration and a move of Spoiler (a zone and an action), the function has_succ_ returns true if the move is allowed for Spoiler, and false otherwise. Moreover, with these parameters together with the reset chosen by Determinizator, the function succs returns the set of successor Configurations. This class is naturally used to define the states of Spoiler and Determinizator.

Class StateOfSpoiler Attributes of this class are a name, a set of Configurations, a zone (DBM) and three booleans: one indicating if the state is losing for Determinizator and two booleans used in the game traversal, one (called willLose) which indicates if it is hopeless for Determinizator to win from it, the other one (called inProgress) to indicate that the traversal from this state is in progress. A StateOfSpoiler has naturally a function to decide whether a pair with a zone and an action constitutes a possible move for Spoiler from this state.

Class StateOfDeterminizator Attributes of this class are a name, a StateOfSpoiler (the predecessor), a zone and an action (the ones chosen by Spoiler), and as for StateOfSpoiler, there is a boolean indicating if the state is losing and two booleans used in the game traversal. A StateOfDeterminizator has a function to compute its successors of the game, some StateOfSpoiler's.

Remark that a StateOfSpoiler v does not need a function to compute the successor because it is simply a StateOfDeterminizator whose predecessor is v, moreover the action and the zone are the label of the move leading to this state (the other attributes are neither the result of a computation). On the other hand, a StateOfDeterminizator does not need a function to compute its possible moves, because they are all allowed.

Class GameOfDeterminization This class contains the main function which builds the game on-the-fly while searching for a winning strategy. Main attributes of this class are a set of StateOfSpoiler's, a set of StateOfDeterminizator's, an initial StateOfSpoiler, tables assigning successors and predecessors to each state, a table assigning to each StateOfDeterminizator a strategic move. The principle of the algorithm can be understood independently on the definition of the game, simply keeping in mind that we want to solve a finite turn-based safety game for player Determinizator. As a consequence, it

suffices to have one safe successor from StateOfDeterminizator to be safe, whereas it suffices to have one unsafe successor from StateOfSpoiler to be unsafe. The program uses three main functions. Let us explain their roles.

- is_safe

  This function applies to a state v of the game either a StateOfDeterminizator, or a StateOfSpoiler. If the marker willLose of the state is equal to true, then the function returns false. If the marker inProgress is equal to true, then the function returns true, otherwise it builds the successors of v.

  If v is a StateOfSpoiler, then the function checks whether v has only safe successors, by calling itself over each of the successors of v. If there is one unsafe successor then Determinizator cannot win from v. The winning strategy that is built on-the-fly is then corrected by calling repair_my_error(v).

  If v is a StateOfDeterminizator, then the function calls has_a_safe_succ(v). If the result is true, then the function returns true, otherwise it has to correct the strategy also by calling repair_my_error(v).

- has_a_safe_succ

  This function applies to a StateOfDeterminizator v. It searches a safe (or rather believed to be safe) successor of v. If it finds one, then it updates the strategy with this choice and returns true, otherwise it returns false.

- repair_my_error

  This function applies to a state v of the game either a StateOfDeterminizator, or a StateOfSpoiler.

  If v is a StateOfSpoiler, the function removes the moves of the strategy leading to v, and for each predecessor v_d, it searches a new safe move by calling has_a_safe_succ(v_d).

  If v is a StateOfDeterminizator, then the function calls itself over all its predecessors. Indeed, every StateOfSpoiler which can lead to v is now known to be unsafe.

The initialization of the program consists in build the initial StateOfSpoiler v_0 and the resolution is performed calling is_safe(v_0).

3.5.3 Execution of the program

Inputs and outputs of our implementation are in *.xml* and they are compatible with UPPAAL. We also defined printing functions for the game in order to observe its structure. Figure 3.5.3 represents a part of a huge game which has been built during our experimentations. States of Spoiler are rectangles and states of Determinizator are circles. Bad states are colored in red. States in gray are the states from which Spoiler can win. The algorithm which we implemented allows in particular to avoid the exploration of useless states. The traversal of the game naturally stops on losing states, but it can also stop on other states when there is no longer hope to win by visiting them. For example, on the top of Figure 3.5.3, we can see that a rectangle is colored in gray because, from it, Spoiler can go in a state of Determinizator which only leads to losing states. Hence, the other successor of this rectangle state has not been explored.

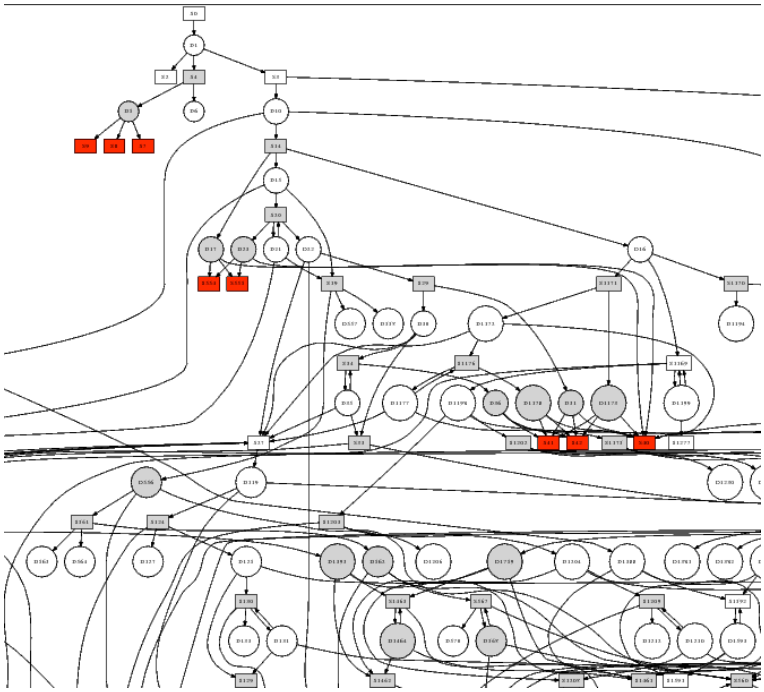


Figure 3.16: Part of a game illustrating the on-the-fly construction of the game.

Experimentations For small examples such as the running example of the chapter, our implementation computes a determinisitic timed automaton in one seconde. In Figure 3.17, the timed automaton on the right is our running example, drawn in UPPAAL in order to be in the format accepted by our program. In the middle of the figure, the game built by our implementation which is drastically smaller than the complete game represented in Figure 3.4 on page 48. Finally, the deterministic resulting timed automaton obtain by our algorithm is the one presented in the chapter. It is also drawn in UPPAAL, thanks to the format of our outputs.

Our implementation also answers in some seconds for quite large examples, provided that there is a single clock. Indeed, experimentations for timed automata with two clocks become very costly. We tried to apply the program to timed automata recognizing MITL formula. When there are two clocks in the example and two clocks for the deterministic timed automaton, the response sometimes comes after one hour of computation, whereas the determinism of the timed automaton is not very complex. This is not surprising because of the doubly exponential complexity of the algorithm with respect to the number of clocks. The main observation of these experimentations is that there is a blow up of the number of states due to the blow up of the number of relations. In this way, the algorithm does, in some sense, several times the same computation. Indeed, very similar states are built, from which the exploration could probably be done only once. These experimentations thus help us to understand more deeply the underlying problem of the determinization. In the future, we aim to imagine heuristics to improve this implementation using our experimentations.

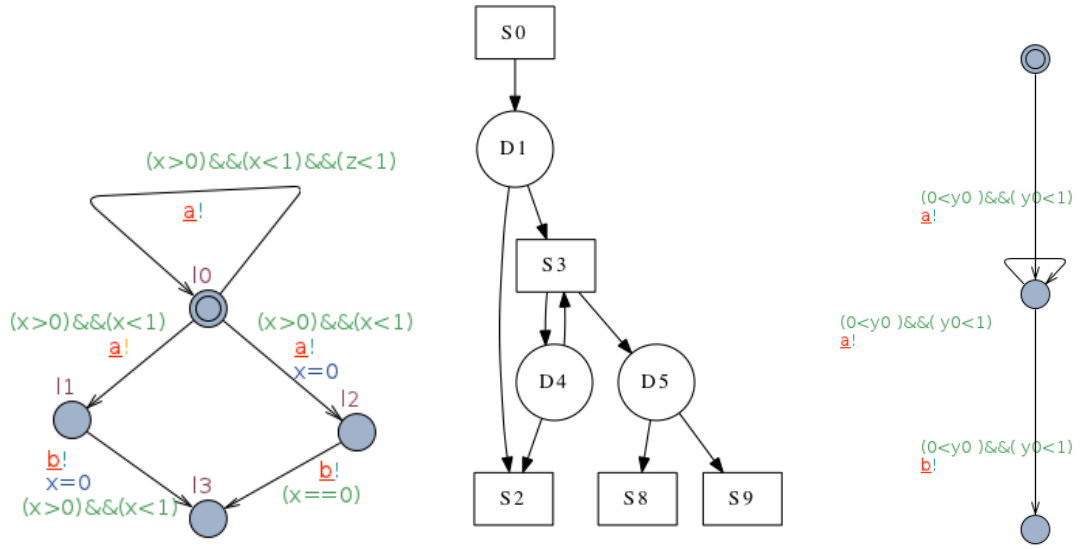


Figure 3.17: Input timed automaton, game and output timed automaton of our running example.

Conclusion

In this chapter, we proposed a game-based approach for the determinization of timed automata. Given a timed automaton $\mathcal{A}$ (with ε-transitions and invariants) and resources (k, M), we build a finite turn-based safety game between two players Spoiler and Determinizator, such that any strategy for Determinizator yields a deterministic over-approximation of the language of $\mathcal{A}$, and any winning strategy provides a deterministic equivalent for $\mathcal{A}$, in both cases, with k clocks and maximal constant M. We also detail how to adapt the framework to generate deterministic under-approximations, or deterministic abstractions combining under- and over-approximations. A motivation for this generalization is to tackle the problem of off-line model-based test generation from non-deterministic timed automata specifications in the next chapter.

Our approach subsumes the two existing methods [KT09, BBBB09]. In comparison with the over-approximation algorithm from [KT09], our game approach yields much more often a deterministic equivalent. In particular, the game approach preserves deterministic timed automata (when sufficient resources are provided), which is not the case for [KT09]. This comes from the fact that strategies can be seen as a generalization of the skeletons of [KT09]: strategies are timed and adaptive, compared to fixed finite-state skeletons. Another interesting point is that our method deals with urgency in a finer way, preserving the invariants as much as possible, whereas the algorithm of [KT09] always over-approximates the urgency status of the transitions as lazy.

Compared to the determinization procedure of [BBBB09], our approach deals with a richer model of timed automata, including ε-transitions and invariants. Already without these extensions, any timed automaton that can be determinized by [BBBB09], can also be determinized by our game-based approach. The class of automatically determinized timed automata is thus strictly increased, thanks to a smoother treatment of relations between the original and the new clocks, and also due to a partial treatment of language inclusion between distinct paths of the original automaton.

The (approximate) determinization of timed automata is a complex problem and the three above

mentioned algorithms run in time doubly exponential in the size of the input. More precisely for our approach, the number of locations of the resulting automaton is doubly exponential in its number of clocks and in the number of clocks of $\mathcal{A}$, and exponential in the number of locations of $\mathcal{A}$. We implemented a prototype tool during a visit in the team of Kim G. Larsen at Aalborg University and in particular thanks to the help of Peter Bulychev. Given the difficulty of the problem, it would be of interest to develop further this prototype by implementing some heuristics. We discuss some related perspectives in the conclusion of the document.

Chapter 4

Application of the Game Approach to Off-line Test Selection

Introduction

In Chapter 3, we introduced a game approach to determinize timed automata. Our algorithm always yields a deterministic timed automaton, but this result can be an approximation. We presented several possible approximations: over-approximation, under-approximation and a mix of them. The main motivation for the latter is the application to off-line test selection which is developed in this chapter.

Conformance testing is the process of testing whether some implementation of a software system behaves correctly with respect to its specification. In this testing framework, implementations are considered as black boxes, *i.e.* the source code is unknown, only their interface with the environment is known and used to interact with the tester. In formal model-based conformance testing, models are used to describe testing artifacts (specifications, implementations, test cases, ...). Moreover, conformance is formally defined as a relation between implementations and specifications which reflects what are the correct behaviors of the implementation with respect to those of the specification. Defining such a relation requires the hypothesis that the implementation behaves as a model. Test cases with verdicts, which will be executed against the implementation in order to check conformance, are generated automatically from the specification. Test generation algorithms should then ensure properties relating verdicts of executions of test cases with the conformance relation (*e.g.* soundness), thus improving the quality of testing compared to manual writing of test cases.

For timed systems, model-based conformance testing has already been explored in the last decade, with different models and conformance relations (see *e.g.* [ST08] for a survey), and various test generation algorithms (e.g. [NS03, BB05, KT09]). In this context, a very popular model is timed automata with inputs and outputs (TAIOs), a variant of timed automata [AD94], in which the alphabet of observable actions is partitioned into inputs and outputs. We consider here a very general model, partially observable and non-deterministic TAIOs with invariants to model urgency. We resort to the **tioco** conformance relation defined for TAIOs [KT04], which is equivalent to the **rtioco** relation [LMN05]. This relation compares the observable behaviors of timed systems, made of inputs, outputs and delays, restricting attention to what happens after specification traces. Intuitively, an implementation conforms to a specification if after any observable trace of the specification, outputs and delays observed on the implementation after this trace are allowed by the specification.

One of the main difficulties encountered in test generation for partially observable, non-deterministic TAIOs is determinization. In fact determinization is required in order to foresee the next enabled ac-

tions during execution, and thus to emit a correct verdict depending on whether actions observed on the implementation are allowed by the specification model after the current observable trace. Unfortunately, as discussed in the introduction of this part, TAs (and thus TAIOs) are not determinizable in general: the class of deterministic TAs is a strict subclass of TAs. Two different approaches have been proposed for test generation from timed models, which induce different treatments of non-determinism.

- In off-line test generation test cases are first generated as timed automata (or timed sequences, or timed transition systems) and subsequently executed on the implementation. One advantage is that test cases can be stored and further used *e.g.* for regression testing and serve for documentation. However, due to the non-determinizability of TAIOs, the approach has often been limited to deterministic or determinizable TAIOs [KJM04, NS03]. A notable exception is [KT09] where the problem is solved by the use of an over-approximate determinization with fixed resources which we reported in the latter chapter. Another one is [DLLN09] where winning strategies of timed games are used as test cases.

- In on-line test generation, test cases are generated during their execution. Given a current observed trace, enabled actions after this trace are computed from the specification model and, either an allowed input is sent to the implementation, or a received output or an observed delay is checked. This technique can be applied to any TAIO, as possible observable actions are computed only along the current finite execution (the set of possible states of the specification model after a finite trace, and their enabled actions are finitely representable and computable), thus avoiding a complete determinization. On-line test generation is of particular interest to rapidly discover errors, and can be applied to large and non-deterministic systems, but it may sometimes be impracticable due to a lack of reactivity: the time needed to compute successor states on-line may sometimes be incompatible with real-time constraints.

In this chapter, we propose to generate test cases off-line for the whole class of non-deterministic TAIOs, in the formal framework of the **tioco** conformance theory. The determinization problem is tackled thanks to the approximate determinization presented in Chapter 3, with fixed resources in the spirit of [KT09]. Our approximate determinization method is more precise than [KT09] (see Chapter 3 for details), preserves the richness of our model by dealing with partial observability (ε-transitions) and urgency (invariants), and is suitable to testing by a different treatment of inputs, outputs and delays (mix of under- and over-approximations). Determinization is exact for known classes of determinizable TAIOs (*e.g.* event-clock TAs, TAs with integer resets, strongly non-Zeno TAs) if resources are sufficient. In the general case, determinization may over-approximate outputs and delays and under-approximate inputs. More precisely, it produces a deterministic io-abstraction of the TAIO for a particular io-refinement relation which generalizes the one of [DLL$^+$10]. As a consequence, if test cases are generated from the io-abstract deterministic TAIO, and are sound for this TAIO, they are guaranteed to be sound for the original non-deterministic TAIO.

Behaviors of specifications to be tested are identified by means of test purposes. Test purposes are often used in testing practice, and are particularly useful when one wants to focus testing on particular behaviors, *e.g.* corresponding to requirements or suspected behaviors of the implementation. In this chapter they are defined as open timed automata with inputs and outputs (OTAIOs), a model generalizing TAIOs, allowing to precisely target some behaviors according to actions and clocks of the specification as well as proper clocks. Then, in the same spirit as for the TGV tool in the untimed case [JJ05], test selection is performed relying on a co-reachability analysis. Produced test cases are TAIOs, while most approaches generate less elaborated test cases, timed traces or trees. In addition

to soundness, when determinization is exact, we also prove an exhaustiveness property, and two other properties on the adequacy of test case verdicts. To our knowledge, this work constitutes the most general and advanced off-line test selection approach for TAIOs.

This chapter is based on the article [BJSK12], a journal version of the paper [BJSK11]. It is structured as follows. In the next section we introduce the model of OTAIOs, its semantics, some notations and operations on this model and the model of TAIOs. Section 4.2 recalls the **tioco** conformance theory for TAIOs, including properties of test cases relating conformance and verdicts, and introduces an io-refinement relation which preserves **tioco**. In Section 4.3 we detail the test selection mechanism using test purposes and prove some properties on generated test cases. Section 4.4 discusses some issues related to test case execution and test purposes and some related work.

4.1 A model of open timed automata with inputs / outputs

In the context of model-based testing, timed automata have been extended to timed automata with inputs and outputs (TAIOs) whose sets of actions are partitioned into inputs, outputs and unobservable actions. In this section, we further extend TAIOs by partitioning the set of clocks into proper clocks (*i.e.*, controlled by the automaton) and observed clocks (*i.e*, owned by some other automaton). The resulting model of *open timed automata with inputs/outputs* (OTAIOs for short), allows one to describe observer timed automata that can test clock values from other automata. While the sub-model of TAIOs (with only proper clocks) is sufficient for most testing artifacts (specifications, implementations, test cases) observed clocks of OTAIOs will be useful to express test purposes whose aim is to focus on the timed behaviors of the specification. Invariants and ε-transitions are respectively used to model urgency of some outputs and internal actions.

4.1.1 Timed automata with inputs/outputs

We start by introducing notations and useful definitions concerning TAIOs and OTAIOs. We write $\sqcup$ for the disjoint union of sets, and use it, when appropriate, to emphasize that sets are disjoint.

Definition 4.1 (OTAIO)**.** *An* open timed automaton with inputs and outputs *(OTAIO) is a tuple* $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{A}}, X_o^{\mathcal{A}}, M^{\mathcal{A}}, \mathsf{Inv}^{\mathcal{A}}, E^{\mathcal{A}})$ *such that:*

- $L^{\mathcal{A}}$ *is a finite set of* locations*, with* $\ell_0^{\mathcal{A}} \in L^{\mathcal{A}}$ *the* initial location*,*
- $\Sigma_?^{\mathcal{A}}$*,* $\Sigma_!^{\mathcal{A}}$ *and* $\Sigma_\tau^{\mathcal{A}}$ *are disjoint finite* alphabets *of* input actions *(noted* $a?, b?, \ldots$*),* output actions *(noted* $a!, b!, \ldots$*), and* internal actions *(noted* $\tau_1, \tau_2, \ldots$*). We note* $\Sigma_{obs}^{\mathcal{A}} = \Sigma_?^{\mathcal{A}} \sqcup \Sigma_!^{\mathcal{A}}$ *for the alphabet of observable actions, and* $\Sigma^{\mathcal{A}} = \Sigma_?^{\mathcal{A}} \sqcup \Sigma_!^{\mathcal{A}} \sqcup \Sigma_\tau^{\mathcal{A}}$ *for the whole set of actions.*
- $X_p^{\mathcal{A}}$ *and* $X_o^{\mathcal{A}}$ *are disjoint finite sets of* proper clocks *and* observed clocks*, respectively. We note* $X^{\mathcal{A}} = X_p^{\mathcal{A}} \sqcup X_o^{\mathcal{A}}$ *for the whole set of* clocks*.*
- $M^{\mathcal{A}} \in \mathbb{N}$ *is the* maximal constant *of* $\mathcal{A}$*, and we will refer to* $(|X^{\mathcal{A}}|, M^{\mathcal{A}})$ *as the* resources *of* $\mathcal{A}$*,*
- $\mathsf{Inv}^{\mathcal{A}} : L^{\mathcal{A}} \to I_{M^{\mathcal{A}}}(X^{\mathcal{A}})$ *is a mapping which labels each location with an* M-bounded invariant*,*
- $E^{\mathcal{A}} \subseteq L^{\mathcal{A}} \times G_{M^{\mathcal{A}}}(X^{\mathcal{A}}) \times \Sigma^{\mathcal{A}} \times 2^{X_p^{\mathcal{A}}} \times L^{\mathcal{A}}$ *is a finite set of* edges *where* guards *are defined on* $X^{\mathcal{A}}$*, but* resets *are restricted to proper clocks in* $X_p^{\mathcal{A}}$*.*

One of the reasons for introducing the OTAIO model is to have a uniform model (syntax and semantics) that will be next specialized for particular testing artifacts. In particular, an OTAIO with an empty set of observed clocks $X_o^{\mathcal{A}}$ is a classical TAIO, and will be the model for specifications, implementations and test cases. The partition of actions reflects their roles in the testing context: the tester cannot observe internal actions, but controls inputs and observes outputs (and delays). The set of clocks is also partitioned into *proper clocks*, *i.e.*usual clocks controlled by the system itself through resets, as opposed to *observed clocks* referring to proper clocks of another OTAIO (*e.g.* modeling the system's environment). These cannot be reset to avoid intrusiveness, but synchronization with them in guards and invariants is allowed. This partition of clocks will be useful for test purposes which can have, as observed clocks, some proper clocks of specifications, with the aim of selecting time constrained behaviors of specifications to be tested.

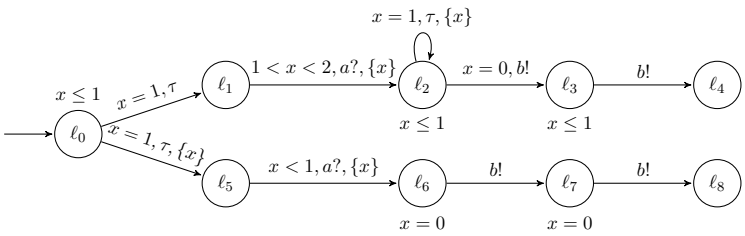


Figure 4.1: Specification $\mathcal{A}$

Running example Figure 4.1 represents a TAIO for a specification $\mathcal{A}$ that will serve as a running example in this paper. Its clocks are $X = X_p^{\mathcal{A}} = \{x\}$, its maximal constant is $M^{\mathcal{A}} = 2$, it has a single input $\Sigma_?^{\mathcal{A}} = \{a\}$, a single output $\Sigma_!^{\mathcal{A}} = \{b\}$ and one internal action $\Sigma_\tau^{\mathcal{A}} = \{\tau\}$. Informally, its behavior is as follows. It may stay in the initial location ℓ_0 while $x \leq 1$, and at $x = 1$, has the choice, either to go to ℓ_1 with action τ, or go to ℓ_5 with action τ while resetting x. In ℓ_1, it may receive a and move to ℓ_2 when x is between 1 and 2, and reset x. In ℓ_2 it may stay while $x \leq 1$ and, either send b and go to ℓ_3 at $x = 0$, or loop silently when $x = 1$ while resetting x. This means that b can be sent at any integer delay after entering ℓ_2. In ℓ_3 it may stay while $x \leq 1$ and move to ℓ_4 when sending b. In ℓ_5, one can move to ℓ_6 before $x = 1$ by receiving a and resetting x. Due to invariants $x = 0$ in ℓ_6 and ℓ_7, the subsequent behavior consists in the immediate transmission of two b's.

4.1.2 The semantics of OTAIOs

Let $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{A}}, X_o^{\mathcal{A}}, M^{\mathcal{A}}, \mathsf{Inv}^{\mathcal{A}}, E^{\mathcal{A}})$ be an OTAIO. The semantics of $\mathcal{A}$ is a *timed transition system* $\mathcal{T}^{\mathcal{A}} = (S^{\mathcal{A}}, s_0^{\mathcal{A}}, \Gamma^{\mathcal{A}}, \rightarrow_{\mathcal{A}})$ where

- $S^{\mathcal{A}} = L^{\mathcal{A}} \times \mathbb{R}_{\geq 0}^{X^{\mathcal{A}}}$ is the set of *states i.e.*pairs (ℓ, v) consisting in a location and a valuation of clocks;
- $s_0^{\mathcal{A}} = (\ell_0^{\mathcal{A}}, \overline{0}) \in S^{\mathcal{A}}$ is the *initial state*;
- $\Gamma^{\mathcal{A}} = \mathbb{R}_{\geq 0} \sqcup E^{\mathcal{A}} \times 2^{X_o^{\mathcal{A}}}$ is the set of transition *labels* consisting in either a delay δ or a pair (e, X_o') formed by an edge $e \in E$ and a set $X_o' \subseteq X_o^{\mathcal{A}}$ of observed clocks;
- the transition relation $\rightarrow_{\mathcal{A}} \subseteq S^{\mathcal{A}} \times \Gamma^{\mathcal{A}} \times S^{\mathcal{A}}$ is the smallest set of the following moves:

– *Discrete moves:* $(\ell, v) \xrightarrow{(e,X'_o)}_{\mathcal{A}} (\ell', v')$ whenever there exists $e = (\ell, g, a, X'_p, \ell') \in E^{\mathcal{A}}$ such that $v \models g \wedge \mathsf{Inv}^{\mathcal{A}}(\ell)$, $X'_o \subseteq X_o^{\mathcal{A}}$ is an arbitrary subset of observed clocks, $v' = v_{[X'_p \sqcup X'_o \leftarrow 0]}$ and $v' \models \mathsf{Inv}^{\mathcal{A}}(\ell')$. Note that X'_o is unconstrained as observed clocks are not controlled by $\mathcal{A}$ but by a peer OTAIO.

– *Time elapse:* $(\ell, v) \xrightarrow{\delta}_{\mathcal{A}} (\ell, v + \delta)$ for $\delta \in \mathbb{R}_{\geq 0}$ if $v + \delta \models \mathsf{Inv}^{\mathcal{A}}(\ell)$.

The semantics of OTAIOs generalizes the usual semantics of TAIOs. The difference lies in the treatment of the additional observed clocks as the evolution of those clocks is controlled by a peer OTAIO. The observed clocks evolve at the same speed as the proper clocks, thus continuous moves are simply extended to proper and observed clocks. For discrete moves however, resets of observed clocks are uncontrolled, thus all possible resets have to be considered.

Let us now fix some vocabulary which can differ from the rest of the document. Indeed, in the testing context, we consider that all the locations are accepting. Runs are note introduced as readers of timed words, but as executions of a systems. We also define sequences and traces of runs, two levels of abstraction of runs.

A *partial run* of $\mathcal{A}$ is a finite sequence of subsequent moves in $(S^{\mathcal{A}} \times \Gamma^{\mathcal{A}})^*.S^{\mathcal{A}}$. For example $\varrho = s_0 \xrightarrow{\delta_1}_{\mathcal{A}} s'_0 \xrightarrow{(e_1,X_o^1)}_{\mathcal{A}} s_1 \cdots s_{k-1} \xrightarrow{\delta_k}_{\mathcal{A}} s'_{k-1} \xrightarrow{(e_k,X_o^k)}_{\mathcal{A}} s_k$. The sum of delays in ϱ is noted $time(\varrho)$. A *run* is a partial run starting in $s_0^{\mathcal{A}}$. A state s is *reachable* if there exists a run leading to s. A state s is *co-reachable* from a set $S' \subseteq S^{\mathcal{A}}$ if there is a partial run from s to a state in S'. We note $\mathtt{reach}(\mathcal{A})$ the set of reachable states and $\mathtt{coreach}(\mathcal{A}, S')$ the set of states co-reachable from S'.

A (partial) *sequence* is a projection of a (partial) run where states are forgotten, and discrete transitions are abstracted to actions and proper resets which are grouped with observed resets. As an example, the sequence corresponding to a run

$$\varrho = s_0 \xrightarrow{\delta_1}_{\mathcal{A}} s'_0 \xrightarrow{(e_1,X_o^1)}_{\mathcal{A}} s_1 \cdots s_{k-1} \xrightarrow{\delta_k}_{\mathcal{A}} s'_{k-1} \xrightarrow{(e_k,X_o^k)}_{\mathcal{A}} s_k$$

is

$$\mu = \delta_1.(a_1, X_p^1 \sqcup X_o^1) \cdots \delta_k.(a_k, X_p^k \sqcup X_o^k)$$

where $e_i = (\ell_i, g_i, a_i, X_p^i, \ell'_i)$ for all $i \in [1, k]$. We then note $s_0 \xrightarrow{\mu}_{\mathcal{A}} s_k$. We write $s_0 \xrightarrow{\mu}_{\mathcal{A}}$ if there exists s_k such that $s_0 \xrightarrow{\mu}_{\mathcal{A}} s_k$. We note $\mathsf{Seq}(\mathcal{A}) \subseteq (\mathbb{R}_{\geq 0} \sqcup (\Sigma^{\mathcal{A}} \times 2^{X^{\mathcal{A}}}))^*$ (respectively $\mathsf{pSeq}(\mathcal{A})$) the set of sequences (resp. partial sequences) of $\mathcal{A}$. For a sequence μ, $time(\mu)$ denotes the sum of delays in μ.

For a (partial) sequence $\mu \in \mathsf{pSeq}(\mathcal{A})$, $Trace(\mu) \in (\mathbb{R}_{\geq 0} \sqcup \Sigma_{obs}^{\mathcal{A}})^*.\mathbb{R}_{\geq 0}$ denotes the observable behavior obtained by erasing internal actions and summing delays between observable ones. It is defined inductively as follows:

- $Trace(\varepsilon) = 0$,
- $Trace(\delta_1 \dots \delta_k) = \Sigma_{i=1}^k \delta_i$,
- $Trace(\delta_1 \dots \delta_k.(\tau, X').\mu) = Trace((\Sigma_{i=1}^k \delta_i).\mu)$,
- $Trace(\delta_1 \dots \delta_k.(a, X').\mu) = (\Sigma_{i=1}^k \delta_i).a.Trace(\mu)$ if $a \in \Sigma_{obs}^{\mathcal{A}}$.

For example $Trace(1.(\tau, X^1).2.(a, X^2).2.(\tau, X^3)) = 3.a.2$ and $Trace(1.(\tau, X^1).2.(a, X^2)) = 3.a.0$. When a trace ends by a 0-delay, we sometimes omit it and write *e.g.* $3.a$ for $3.a.0$.

When concatenating two traces, the last delay of the first trace and the initial delay of the second one must be added up as follows: if $\sigma_1 = \delta_1.a_1.\cdots a_n.\delta_{n+1}$ and $\sigma_2 = \delta'_1.a'_1.\cdots a'_m.\delta'_{m+1}$ then $\sigma_1.\sigma_2 = \delta_1.a_1.\cdots a_n.(\delta_{n+1} + \delta'_1).a'_1.\cdots a'_m.\delta'_{m+1}$. Concatenation allows one to define the notion of prefix. Given a trace σ, σ_1 is a *prefix* of σ if there exists some σ_2 with $\sigma = \sigma_1.\sigma_2$. Under this definition, $1.a.1$ is a prefix of $1.a.2.b$.

For a run ϱ projecting onto a sequence μ, we also write $Trace(\varrho)$ for $Trace(\mu)$. The set of traces of runs of $\mathcal{A}$ is denoted by $\mathsf{Traces}(\mathcal{A}) \subseteq (\mathbb{R}_{\geq 0} \sqcup \Sigma^{\mathcal{A}}_{obs})^*.\mathbb{R}_{\geq 0}$ [1].

Two OTAIOs are said *equivalent* if they have the same sets of traces.

Let $\sigma \in (\mathbb{R}_{\geq 0} \sqcup \Sigma^{\mathcal{A}}_{obs})^*.\mathbb{R}_{\geq 0}$ be a trace, and $s \in S^{\mathcal{A}}$ be a state,

- $\mathcal{A}$ `after` $\sigma = \{s \in S^{\mathcal{A}} \mid \exists \mu \in \mathsf{Seq}(\mathcal{A}), s^{\mathcal{A}}_0 \xrightarrow{\mu}_{\mathcal{A}} s \wedge Trace(\mu) = \sigma\}$ denotes the set of states where $\mathcal{A}$ can stay after observing the trace σ.
- $elapse(s) = \{t \in \mathbb{R}_{\geq 0} \mid \exists \mu \in (\mathbb{R}_{\geq 0} \sqcup (\Sigma^{\mathcal{A}}_{\tau} \times 2^{X^{\mathcal{A}}}))^*, s \xrightarrow{\mu}_{\mathcal{A}} \wedge time(\mu) = t\}$ is the set of enabled delays in s with no observable action.
- $out(s) = \{a \in \Sigma^{\mathcal{A}}_{!} \mid \exists X \subseteq X^{\mathcal{A}}, s \xrightarrow{(a,X)}_{\mathcal{A}}\} \cup elapse(s)$ (and $in(s) = \{a \in \Sigma^{\mathcal{A}}_{?} \mid \exists X \subseteq X^{\mathcal{A}}, s \xrightarrow{(a,X)}_{\mathcal{A}}\}$) for the set of outputs and delays (respectively inputs) that can be observed from s. For $S' \subseteq S^{\mathcal{A}}$, $out(S') = \bigcup_{s \in S'} out(s)$ and $in(S') = \bigcup_{s \in S'} in(s)$.

Using these last definitions, we will later describe the set of possible outputs and delays after the trace σ by $out(\mathcal{A}$ `after` $\sigma)$.

Notice that all notions introduced for OTAIOs apply to the subclass of TAIOs.

4.1.3 Properties and operations

A TAIO $\mathcal{A}$ is *deterministic* (and called a DTAIO) whenever for any $\sigma \in \mathsf{Traces}(\mathcal{A})$, $\mathcal{A}$ `after` σ is a singleton[2]. A TAIO $\mathcal{A}$ is *determinizable* if there exists an equivalent DTAIO.

An OTAIO $\mathcal{A}$ is said *complete* if in every location ℓ, $\mathsf{Inv}^{\mathcal{A}}(\ell) =$ `true` and for every action $a \in \Sigma^{\mathcal{A}}$, the disjunction of all guards of transitions leaving ℓ and labeled by a is `true`. This entails that $\mathsf{Seq}(\mathcal{A}) \downarrow_{X^{\mathcal{A}}_p} = (\mathbb{R}_{\geq 0} \sqcup (\Sigma^{\mathcal{A}} \times 2^{X^{\mathcal{A}}_o}))^*$, where $\downarrow_{X^{\mathcal{A}}_p}$ is the projection that removes resets of proper clocks in $X^{\mathcal{A}}_p$. This means that $\mathcal{A}$ is universal for all the behaviors of its environment.

An OTAIO $\mathcal{A}$ is *input-complete* in a state $s \in$ `reach`$(\mathcal{A})$, if $in(s) = \Sigma^{\mathcal{A}}_{?}$. An OTAIO $\mathcal{A}$ is input-complete if it is input-complete in all its reachable states.

An OTAIO $\mathcal{A}$ is *non-blocking* if $\forall s \in$ `reach`$(\mathcal{A}), \forall t \in \mathbb{R}_{\geq 0}, \exists \mu \in \mathsf{pSeq}(\mathcal{A}) \cap (\mathbb{R}_{\geq 0} \sqcup ((\Sigma^{\mathcal{A}}_{!} \sqcup \Sigma^{\mathcal{A}}_{\tau}) \times 2^{X^{\mathcal{A}}}))^*, time(\mu) = t \wedge s \xrightarrow{\mu}_{\mathcal{A}}$. This means that it never blocks the evolution of time, waiting for an input.

For modeling the behavior of composed systems, in particular for modeling the execution of test cases on implementations, we introduce the classical parallel product. This operation consists in the synchronization of two TAIOs on complementary observable actions (*e.g.* $a!$, the emission of a and $a?$ its reception) and induces the intersection of the sets of traces. It is only defined for *compatible* TAIOs, *i.e.* $\mathcal{A}^i = (L^i, \ell^i_0, \Sigma^i_?, \Sigma^i_!, \Sigma^i_\tau, X^i_p, M^i, \mathsf{Inv}^i, E^i)$ for $i = 1, 2$ such that $\Sigma^1_! = \Sigma^2_?$, $\Sigma^1_? = \Sigma^2_!$, $\Sigma^1_\tau \cap \Sigma^2_\tau = \emptyset$ and $X^1_p \cap X^2_p = \emptyset$.

[1] Notice that formally, a trace always ends with a delay, which can be 0. This technical detail is useful later to define verdicts as soon as possible without waiting for a hypothetical next action.

[2] Determinism is only defined (and used in the sequel) for TAIOs. For OTAIOs, the right definition would consider the projection of $\mathcal{A}$ `after` σ which forgets values of observed clocks, as these introduce "environmental" non-determinism.

Definition 4.2 (Parallel product)**.** *The* parallel product *of two compatible TAIOs* $\mathcal{A}^i = (L^i, \ell_0^i, \Sigma_?^i, \Sigma_!^i, \Sigma_\tau^i, X_p^i, M^i, \mathsf{Inv}^i, E^i)$ $i = 1, 2$ *is a TAIO* $\mathcal{A}^1 \| \mathcal{A}^2 = (L, \ell_0, \Sigma_?, \Sigma_!, \Sigma_\tau, X_p, M, \mathsf{Inv}, E)$ *where:*

- $L = L^1 \times L^2$, $\ell_0 = (\ell_0^1, \ell_0^2)$,
- $\Sigma_? = \Sigma_?^1$, $\Sigma_! = \Sigma_!^1$ *and* $\Sigma_\tau = \Sigma_\tau^1 \sqcup \Sigma_\tau^2$
- $X_p = X_p^1 \sqcup X_p^2$
- $M = \max(M^1, M^2)$
- $\forall (\ell^1, \ell^2) \in L, \mathsf{Inv}((\ell^1, \ell^2)) = \mathsf{Inv}(\ell^1) \wedge \mathsf{Inv}(\ell^2)$
- E *is the smallest relation such that:*
  - *for* $a \in \Sigma_?^1 \sqcup \Sigma_!^1$, *if* $(\ell^1, g^1, a, X_p'^1, \ell'^1) \in E^1$ *and* $(\ell^2, g^2, a, X_p'^2, \ell'^2) \in E^2$ *then* $((\ell^1, \ell^2), g^1 \wedge g^2, a, X_p'^1 \cup X_p'^2, (\ell'^1, \ell'^2)) \in E$, i.e.*complementary actions synchronize, corresponding to a communication;*
  - *for* $\tau_1 \in \Sigma_\tau^1$, $\ell^2 \in L^2$, *if* $(\ell^1, g^1, \tau_1, X_p'^1, \ell'^1) \in E^1$ *then* $((\ell^1, \ell^2), g^1, \tau_1, X_p'^1, (\ell'^1, \ell^2)) \in E$, i.e.*internal actions of* $\mathcal{A}_1$ *progress independently;*
  - *for* $\tau_2 \in \Sigma_\tau^2$, $\ell^1 \in L^1$, *if* $(\ell^2, g^2, \tau_2, X_p'^2, \ell'^2) \in E^2$ *then* $((\ell^1, \ell^2), g^2, \tau_2, X_p'^2, (\ell^1, \ell'^2)) \in E$, i.e. *internal actions of* $\mathcal{A}_2$ *progress independently.*

By the definition of the transition relation E of $\mathcal{A}^1 \| \mathcal{A}^2$, TAIOs synchronize exactly on complementary observable actions and time, and evolve independently on internal actions. As a consequence, the following equality on traces holds:

$$\mathsf{Traces}(\mathcal{A}^1 \| \mathcal{A}^2) = \mathsf{Traces}(\mathcal{A}^1) \cap \mathsf{Traces}(\mathcal{A}^2) \tag{4.1}$$

Notice that the definition is not absolutely symmetrical, as the direction (input/output) of actions of the product is chosen with respect to $\mathcal{A}^1$. The technical reason is that, in the execution of a test case on an implementation, we will need to keep the directions of actions of the implementation.

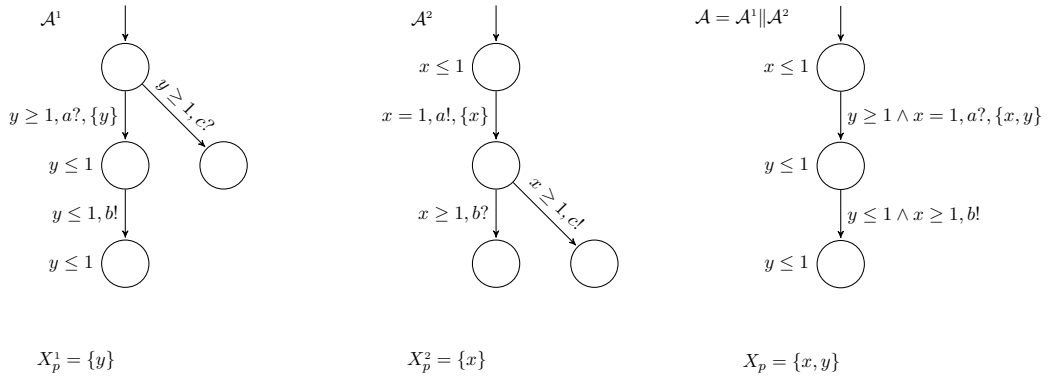


Figure 4.2: Example of a parallel product $\mathcal{A} = \mathcal{A}^1 \| \mathcal{A}^2$.

The Figure 4.2 gives a very simple illustration of the parallel product. The intersection of the sets of traces is clear. Indeed, the parallel product recognizes exactly all prefixes of the trace $1.a.1.b$.

We now define a product operation on OTAIOs which extends the classical product of TAs, with a particular attention to observed clocks. This product is used later in the paper, to model the action of a test purpose which observes the clocks of a specification.

Definition 4.3 (Product)**.** *Let $\mathcal{A}^i = (L^i, \ell_0^i, \Sigma_?, \Sigma_!, \Sigma_\tau, X_p^i, X_o^i, M^i, \mathsf{Inv}^i, E^i)$, $i = 1, 2$, be two OTAIOs with same alphabets and disjoint sets of proper clocks ($X_p^1 \cap X_p^2 = \emptyset$). Their* product *is the OTAIO $\mathcal{A}^1 \times \mathcal{A}^2 = (L, \ell_0, \Sigma_?, \Sigma_!, \Sigma_\tau, X_p, X_o, M, \mathsf{Inv}, E)$ where:*

- $L = L^1 \times L^2$;
- $\ell_0 = (\ell_0^1, \ell_0^2)$;
- $X_p = X_p^1 \sqcup X_p^2$, $X_o = (X_o^1 \cup X_o^2) \setminus X_p$;
- $M = \max(M^1, M^2)$;
- $\forall (\ell^1, \ell^2) \in L, \mathsf{Inv}((\ell^1, \ell^2)) = \mathsf{Inv}^1(\ell^1) \wedge \mathsf{Inv}^2(\ell^2)$;
- $((\ell^1, \ell^2), g^1 \wedge g^2, a, X_p'^1 \sqcup X_p'^2, (\ell'^1, \ell'^2)) \in E$ *if* $(\ell^i, g^i, a, X_p'^i, \ell'^i) \in E^i$, *i=1,2.*

Intuitively, $\mathcal{A}^1$ and $\mathcal{A}^2$ synchronize on both time and common actions (including internal ones[3]). $\mathcal{A}^2$ may observe proper clocks of $\mathcal{A}^1$ using its observed clocks $X_p^1 \cap X_o^2$, and *vice versa*. The set of proper clocks of $\mathcal{A}^1 \times A^2$ is the union of proper clocks of $\mathcal{A}^1$ and A^2, and observed clocks of $\mathcal{A}^1 \times A^2$ are observed clocks of any OTAIO which are not proper. For example, the OTAIO in Figure 4.8 represents the product of the TAIO $\mathcal{A}$ in Figure 4.1 and the OTAIO $\mathcal{TP}$ of Figure 4.7.

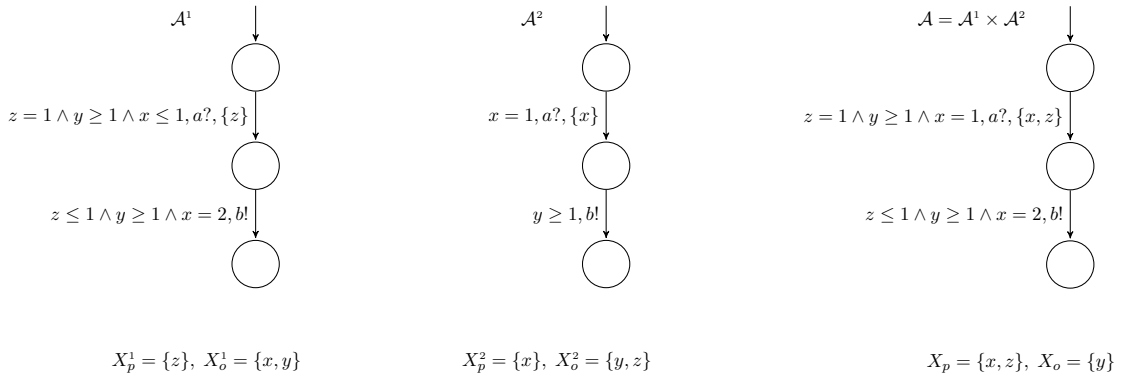


Figure 4.3: Example of a product $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2$.

Contrary to the parallel product, the set of traces of the product of two OTAIOs is not the intersection of the sets of traces of these TAIOs, as illustrated by the following example.

[3]Synchronizing internal actions allows for more precision in test selection. This justifies to have a set of internal actions in the TAIO model.

Illustration of the product Figure 4.3 artificially illustrates the notion of product of two OTAIOs. One can see that $1.a?.1.b!$ is a trace of $\mathcal{A}^1$ and $\mathcal{A}^2$ but is not a trace of $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2$. Indeed, in $\mathcal{A}^1$, $1.a?.1.b!$ is the trace of a sequence where x is not reset at the first action. Unfortunately, the clock x is observed by $\mathcal{A}^1$ but is a proper clock of $\mathcal{A}^2$ which resets it at the first action. As a consequence, $1.a?.1.b!$ cannot be a trace of the product $\mathcal{A}^1 \times \mathcal{A}^2$. In fact, the second edge in $\mathcal{A}$ can never be fired, since clocks z and x agree on their values and cannot be simultaneously smaller than 1 and equal to 2.

On the other hand, sequences are more adapted to express the underlying operation. To compare the sets of sequences of $\mathcal{A}^1 \times \mathcal{A}^2$ with the sets of sequences of its factors, we introduce an operation that lifts the sets of clocks of factors to the set of clocks of the product: for $\mathcal{A}^1$ defined on (X_p^1, X_o^1), and $X_p^1 \cap X_p^2 = \emptyset$, $\mathcal{A}^1 \uparrow^{(X_p^2, X_o^2)}$ denotes an automaton identical to $\mathcal{A}^1$ but defined on $(X_p^1, X_p^2 \cup X_o^1 \cup X_o^2 \setminus X_p^1)$. The effect on the semantics is to duplicate moves of $\mathcal{A}^1$ with unconstrained resets in $(X_p^2 \cup X_o^2) \setminus (X_p^1 \cup X_o^1)$, so that $\mathcal{A}^1 \uparrow^{(X_p^2, X_o^2)}$ strongly bisimulates $\mathcal{A}^1$. The equivalence just consists in ignoring values of added clocks which do not interfere in the guards. Similarly $\mathcal{A}^2 \uparrow^{(X_p^1, X_o^1)}$ is defined on $(X_p^2, X_p^1 \cup X_o^2 \cup X_o^1 \setminus X_p^2)$. Both $\mathcal{A}^1 \uparrow^{X_p^2, X_o^2}$ and $\mathcal{A}^2 \uparrow^{X_p^1, X_o^1}$ have sequences in $(\mathbb{R}_{\geq 0} \sqcup (\Sigma_\tau^{\mathcal{A}} \times (X_p^1 \cup X_p^2 \cup X_o^1 \cup X_o^2)))^*$. They synchronize on both delays and common actions with their resets. The effect of the product is to restrict the respective environments (observed clocks) by imposing the resets of the peer TAIO. The sequences of the product are then characterized by

$$\mathsf{Seq}(\mathcal{A}^1 \times \mathcal{A}^2) = \mathsf{Seq}(\mathcal{A}^1 \uparrow^{(X_p^2, X_o^2)}) \cap \mathsf{Seq}(\mathcal{A}^2 \uparrow^{(X_p^1, X_o^1)}) \tag{4.2}$$

meaning that the product of OTAIOs is the adequate operation for intersecting sets of sequences.

An OTAIO equipped with a set of states $F \subseteq S^{\mathcal{A}}$ can play the role of an acceptor. A run is *accepted* in F if it ends in F. $\mathsf{Seq}_F(\mathcal{A})$ denotes the set of sequences of accepted runs and $\mathsf{Traces}_F(\mathcal{A})$ the set of their traces. By abuse of notation, if L is a subset of locations in $L^{\mathcal{A}}$, we note $\mathsf{Seq}_L(\mathcal{A})$ for $\mathsf{Seq}_{L \times \mathbb{R}_{\geq 0}^{X^{\mathcal{A}}}}(\mathcal{A})$ and similarly for $\mathsf{Traces}_L(\mathcal{A})$. Note that for the product $\mathcal{A}^1 \times \mathcal{A}^2$, if F^1 and F^2 are subsets of states of $\mathcal{A}^1$ and $\mathcal{A}^2$ respectively, additionally to (4.2), the following equality holds:

$$\mathsf{Seq}_{F^1 \times F^2}(\mathcal{A}^1 \times \mathcal{A}^2) = \mathsf{Seq}_{F^1}(\mathcal{A}^1 \uparrow^{(X_p^2, X_o^2)}) \cap \mathsf{Seq}_{F^2}(\mathcal{A}^2 \uparrow^{(X_p^1, X_o^1)}). \tag{4.3}$$

4.2 Conformance testing theory

In this section, we recall the conformance theory for timed automata based on the conformance relation **tioco** [KT09] that formally defines the set of correct implementations of a given TAIO specification. **tioco** is a natural extension of the **ioco** relation of Tretmans [Tre96] to timed systems. We then define test cases, formalize their executions, verdicts and expected properties relating verdicts to conformance. Finally, we introduce a refinement relation between TAIOs that preserves **tioco**, and will be useful in proving test case properties.

4.2.1 The tioco conformance theory

We consider that the specification is given as a (possibly non-deterministic) TAIO $\mathcal{A}$. The implementation is a black box, unknown except for its alphabet of observable actions, which is the same as the one of $\mathcal{A}$. As usual, in order to formally reason about conformance, we assume that the implementation can be modeled by an (unknown) TAIO.

Definition 4.4 (Implementation)**.** *Let $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{A}}, \emptyset, M^{\mathcal{A}}, \mathsf{Inv}^{\mathcal{A}}, E^{\mathcal{A}})$ be a specification TAIO. An implementation of $\mathcal{A}$ is an input-complete and non-blocking TAIO $\mathcal{I} = (L^{\mathcal{I}}, \ell_0^{\mathcal{I}}, \Sigma_?,$*

$\Sigma_!, \Sigma_\tau^{\mathcal{I}}, X_p^{\mathcal{I}}, \emptyset, M^{\mathcal{I}}, \mathsf{Inv}^{\mathcal{I}}, E^{\mathcal{I}})$ *with same observable alphabet as* $\mathcal{A}$ *(*$\Sigma_?^{\mathcal{I}} = \Sigma_?^{\mathcal{A}}$ *and* $\Sigma_!^{\mathcal{I}} = \Sigma_!^{\mathcal{A}}$*).* $\mathcal{I}(\mathcal{A})$ *denotes the set of possible implementations of* $\mathcal{A}$*.*

The requirements that an implementation is input-complete and non-blocking will ensure that the execution of a test case on $\mathcal{I}$ does not block before verdicts are emitted.

Among the possible implementations in $\mathcal{I}(\mathcal{A})$, the conformance relation **tioco** (for *timed input-output conformance*) [KT09] formally defines which ones conform to $\mathcal{A}$, naturally extending the classical **ioco** relation of Tretmans [Tre96] to timed systems.

Definition 4.5 (Conformance relation)**.** *Let* $\mathcal{A}$ *be a TAIO representing the specification and* $\mathcal{I} \in \mathcal{I}(\mathcal{A})$ *be an implementation of* $\mathcal{A}$*. We say that* $\mathcal{I}$ *conforms to* $\mathcal{A}$ *and write* $\mathcal{I}$ ***tioco*** $\mathcal{A}$ *if* $\forall \sigma \in \mathsf{Traces}(\mathcal{A}), out(\mathcal{I}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{A}\ \texttt{after}\ \sigma)$.

Note that **tioco** is equivalent to the **rtioco** relation that was defined independently in [LMN05] (see [ST08]). Intuitively, $\mathcal{I}$ conforms to $\mathcal{A}$ if after any timed trace enabled in $\mathcal{A}$, every output or delay of $\mathcal{I}$ is specified in $\mathcal{A}$. This means that $\mathcal{I}$ may accept more inputs than $\mathcal{A}$, but is authorized to send less outputs, or send them during a more restricted time interval.

Illustration of the notion of conformance relation Figure 4.4 represents a specification $\mathcal{A}$ and two possible implementations $\mathcal{I}_1$ and $\mathcal{I}_2$. Note that $\mathcal{I}_1$ and $\mathcal{I}_2$ should be input-complete, but for simplicity of figures, we omit some inputs and consider that missing inputs loop to the current location. It is easy to see that $\mathcal{I}_1$ conforms to $\mathcal{A}$. Indeed, it accepts more inputs, which is allowed (after the trace ϵ, $\mathcal{I}_1$ can receive a and d while $\mathcal{A}$ only accepts a), and emits the output b during a more restricted interval of time ($out(\mathcal{I}_1\ \texttt{after}\ a.2) = [0,\infty)$ is included in $out(\mathcal{A}\ \texttt{after}\ a.2) = [0,\infty) \sqcup \{b\}$). On the other hand $\mathcal{I}_2$ does not conform to $\mathcal{A}$ for two reasons: $\mathcal{I}_2$ may send a new output c and may send b during a larger time interval (*e.g.* $out(\mathcal{I}_2\ \texttt{after}\ a.1) = [0,\infty) \sqcup \{b,c\}$ is not included in $out(\mathcal{A}\ \texttt{after}\ a.1) = [0,\infty)$).

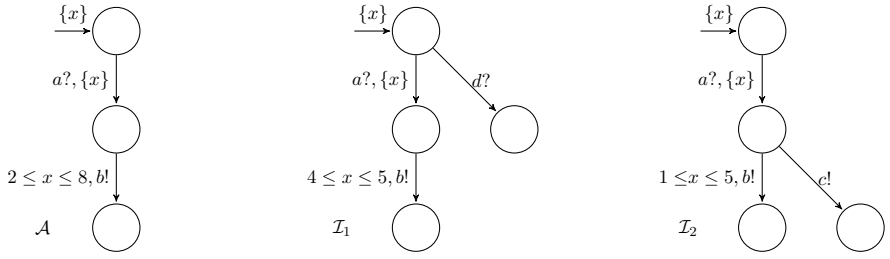


Figure 4.4: Example of a specification $\mathcal{A}$ and two implementations $\mathcal{I}_1$ and $\mathcal{I}_2$.

In practice, conformance is checked by test cases run on implementations. In our setting, we define test cases as deterministic TAIOs equipped with verdicts defined by a partition of states.

Definition 4.6 (Test suite, test case)**.** *Given a specification TAIO* $\mathcal{A}$*, a* test suite *is a set of* test cases*, where a* test case *is a pair* $(\mathcal{TC}, \textbf{Verdicts})$ *consisting of:*

- *a deterministic TAIO* $\mathcal{TC} = (L^{\mathcal{TC}}, \ell_0^{\mathcal{TC}}, \Sigma_?^{\mathcal{TC}}, \Sigma_!^{\mathcal{TC}}, \emptyset, X_p^{\mathcal{TC}}, \emptyset, M^{\mathcal{TC}}, \mathsf{Inv}^{\mathcal{TC}}, E^{\mathcal{TC}})$*,*
- *a partition* **Verdicts** *of the set of states* $S^{\mathcal{TC}} =$ **None** $\sqcup$ **Inconc** $\sqcup$ **Pass** $\sqcup$ **Fail**. *States outside* **None** *are called* verdict states*.*

We also require that

- $\Sigma_?^{\mathcal{TC}} = \Sigma_!^{\mathcal{A}}$ *and* $\Sigma_!^{\mathcal{TC}} = \Sigma_?^{\mathcal{A}}$,
- $\mathcal{TC}$ *is non-blocking, (*e.g. $\mathsf{Inv}^{\mathcal{TC}}(\ell) = \texttt{true}$ *for all* $\ell \in L^{\mathcal{TC}}$*),*
- $\mathcal{TC}$ *is input-complete in all* **None** *states, meaning that it is ready to receive any input from the implementation before reaching a verdict.*

In the following, for simplicity we will sometimes abuse notations and simply write $\mathcal{TC}$ for $(\mathcal{TC}, \textbf{Verdicts})$. Let us give some intuition about the different verdicts of test cases. **Fail** states are those where the test case rejects an implementation. The intention is thus to detect a non-conformance. **Pass** and **Inconc** states are linked to test purposes (see Section 4.3): the intention is that **Pass** states should be those where no non-conformance has been detected and the test purpose is satisfied, whereas **Inconc** states should be those states where no non-conformance has been detected, but the test purpose cannot be satisfied anymore. **None** states are all other states. We insist on the fact that those are intentional characterizations of the verdicts. Properties of test cases defined later specify whether these intentions are satisfied by test cases. We will see that it is not always the case for all properties.

The execution of a test case $\mathcal{TC} \in Test(\mathcal{A})$ on an implementation $\mathcal{I} \in \mathcal{I}(\mathcal{A})$ is modeled by the parallel product $\mathcal{I} \| \mathcal{TC}$, which entails that $\mathsf{Traces}(\mathcal{I} \| \mathcal{TC}) = \mathsf{Traces}(\mathcal{I}) \cap \mathsf{Traces}(\mathcal{TC})$. The facts that $\mathcal{TC}$ is input-complete (in **None** states) and non-blocking while $\mathcal{I}$ is input-complete (in all states) and non-blocking ensure that no deadlock occurs before a verdict is reached.

We say that the verdict of an execution of trace $\sigma \in \mathsf{Traces}(\mathcal{TC})$, noted $\texttt{Verdict}(\sigma, \mathcal{TC})$, is **Pass**, **Fail**, **Inconc** or **None** if $\mathcal{TC}$ `after` σ is included in the corresponding states set [4]. We write $\mathcal{I}$ `fails` $\mathcal{TC}$ if some execution σ of $\mathcal{I} \| \mathcal{TC}$ leads $\mathcal{TC}$ to a **Fail** state, *i.e.*when $\mathsf{Traces}_{\mathbf{Fail}}(\mathcal{TC}) \cap \mathsf{Traces}(\mathcal{I}) \neq \emptyset$, which means that there exists $\sigma \in \mathsf{Traces}(\mathcal{I}) \cap \mathsf{Traces}(\mathcal{TC})$ such that $\texttt{Verdict}(\sigma, \mathcal{TC}) = \textbf{Fail}$. Notice that this is only a possibility to reach the **Fail** verdict among the infinite set of executions of $\mathcal{I} \| \mathcal{TC}$. Hitting one of these executions is not ensured both because of the lack of control of $\mathcal{TC}$ on $\mathcal{I}$ and of timing constraints imposed by these executions.

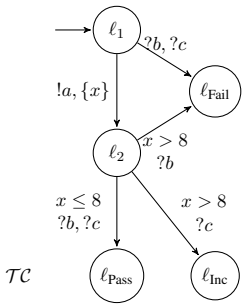
We now introduce soundness, a crucial property ensured by our test generation method. We also introduce exhaustiveness and strictness that will be ensured when determinization is exact (see Section 4.3).

Definition 4.7 (Test suite soundness, exhaustiveness and strictness)**.** *A test suite* $\mathcal{TS}$ *for* $\mathcal{A}$ *is:*

- sound *if* $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\forall \mathcal{TC} \in \mathcal{TS}$, $\mathcal{I}$ `fails` $\mathcal{TC} \Rightarrow \neg(\mathcal{I}$ ***tioco*** $\mathcal{A})$,
- exhaustive *if* $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\neg(\mathcal{I}$ ***tioco*** $\mathcal{A}) \Rightarrow \exists \mathcal{TC} \in \mathcal{TS}$, $\mathcal{I}$ `fails` $\mathcal{TC}$,
- strict *if* $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A}), \forall \mathcal{TC} \in \mathcal{TS}, \neg(\mathcal{I} \| \mathcal{TC}$ ***tioco*** $\mathcal{A}) \Rightarrow \mathcal{I}$ `fails` $\mathcal{TC}$.

Intuitively, soundness means that no conformant implementation can be rejected by the test suite, *i.e.*any failure of a test case during its execution characterizes a non-conformance. Conversely, exhaustiveness means that every non-conformant implementation may be rejected by the test suite. Remember that the definition of $\mathcal{I}$ `fails` $\mathcal{TC}$ indicates only a possibility of reject. Finally, strictness means that non-conformance is detected once it occurs. In fact, $\neg(\mathcal{I} \| \mathcal{TC}$ **tioco** $\mathcal{A})$ means that there is a trace common to $\mathcal{TC}$ and $\mathcal{I}$ which does not conform to $\mathcal{A}$. The universal quantification on $\mathcal{I}$ and $\mathcal{TC}$ implies that any such trace will fail $\mathcal{TC}$. In particular, this implies that failure will be detected as soon as it occurs.

[4] Note that TC being deterministic, $\mathcal{TC}$ `after` σ is a singleton.



$$
\begin{array}{lcl}
\textbf{None} & = & \{\ell_1\} \times \mathbb{R}_{\geq 0} \cup \{\ell_2\} \times [0, 8] \\
\textbf{Inconc} & = & \{\ell_{\text{Inc}}, \ell_2\} \times (8, \infty) \\
\textbf{Pass} & = & \{\ell_{\text{Pass}}\} \times \mathbb{R}_{\geq 0} \\
\textbf{Fail} & = & \{\ell_{\text{Fail}}\} \times \mathbb{R}_{\geq 0}
\end{array}
$$

$$\mathcal{TS} = \{\mathcal{TC}\}$$

Figure 4.5: Example of a sound but not strict test suite for the specification $\mathcal{A}$ (Figure 4.4).

Illustration of the notion of soundness Figure 4.5 represents a test suite composed of a single test case $\mathcal{TC}$ for the specification $\mathcal{A}$ of the Figure 4.4. Indeed, $\mathcal{TC}$ is a TAIO which is input-complete in the **None** states. $\mathcal{TS}$ is sound because the **Fail** states of $\mathcal{TC}$ are reached only when a conformance error occurs, *e.g.* on trace $1.b$. However, this test case can observe non-conformant traces without detecting them, hence $\mathcal{TS}$ is not strict. For example, $1.a.1.b$, $1.a.1.c$ and $1.a.9.c$ are non-conformant traces that do not imply a **Fail** verdict. These traces are *e.g.* traces of $\mathcal{I}_2$ (Figure 4.4) which should allow to detect that $\neg(\mathcal{I}_2$ **tioco** $\mathcal{A})$.

4.2.2 Refinement preserving tioco

In the previous chapter, on page 68, we introduce a refinement relation for two-alphabet timed automata. In this chapter, we use this refinement with the partition input-output of the alphabet. In the testing context, we called it io-refinement relation and give an equivalent definition based on traces. Informally $\mathcal{A}$ io-refines $\mathcal{B}$ if $\mathcal{A}$ specifies more inputs and allows less outputs and delays. As a consequence, if $\mathcal{A}$ and $\mathcal{B}$ are specifications, $\mathcal{A}$ is more restrictive than $\mathcal{B}$ with respect to conformance. We thus prove that io-abstraction (the inverse relation) preserves **tioco**: if $\mathcal{I}$ conforms to $\mathcal{A}$, it also conforms to any io-abstraction $\mathcal{B}$ of $\mathcal{A}$. This will ensure that soundness of test cases is preserved by the approximate determinization defined in Chapter 3.

Definition 4.8. *Let $\mathcal{A}$ and $\mathcal{B}$ be two TAIOs with same input and output alphabets, we say that $\mathcal{A}$* io-refines *$\mathcal{B}$ (or $\mathcal{B}$* io-abstracts *$\mathcal{A}$) and note $\mathcal{A} \preceq \mathcal{B}$ if*

$$
\begin{array}{ll}
(i) & \forall \sigma \in \mathsf{Traces}(\mathcal{B}),\ out(\mathcal{A}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{B}\ \texttt{after}\ \sigma) \text{ and,} \\
(ii) & \forall \sigma \in \mathsf{Traces}(\mathcal{A}),\ in(\mathcal{B}\ \texttt{after}\ \sigma) \subseteq in(\mathcal{A}\ \texttt{after}\ \sigma).
\end{array}
$$

If $\mathcal{B}$ has no ε-transition, this definition is equivalent to the definition of the refinement relation for two-alphabet timed automata fixing the first alphabet as the set of the inputs and the second one as the set of outputs. The second property of this definition is equivalent to the last property of the (Σ_1, Σ_2)-refinement. Indeed, recall the considered property: if $s'_0 \xRightarrow{w.(t,a_1)}_{\mathcal{B}}$ where $a_1 \in \Sigma_1$, and $s_0 \xRightarrow{w}_{\mathcal{A}} \xrightarrow{\tau_0}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \xrightarrow{\tau_n}_{\mathcal{A}}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$ and the accumulated delay of w is $t - \sum_{i=0}^{n} \tau_i$ then $s_0 \xRightarrow{w.(t,a_1)}_{\mathcal{A}}$. Such a property can be expressed in term of traces to obtain the above definition. Indeed, notting $\sigma.\tau.a_1$ for the sequence of delays and observable actions corresponding to this word $w.(t, a_1)$, $s'_0 \xRightarrow{w.(t,a_1)}_{\mathcal{B}}$ implies that the trace $\sigma.\tau$ is a trace of $\mathcal{B}$ and if a_1 is an input, then $a_1 \in in(\mathcal{B}\ \texttt{after}\ \sigma.\tau)$. then the condition over $\mathcal{A}$ means that $\sigma.\tau$ is also a trace of $\mathcal{A}$ and the conclusion means that $a_1 \in in(\mathcal{A}\ \texttt{after}\ \sigma.\tau)$. The out function returns enable output actions and

delays, the first property of this definition is thus equivalent to the conjunction of both first properties of the (Σ_1, Σ_2)-refinement.

As we will see below, $\preceq$ is a preorder relation. Moreover, as condition (ii) is always satisfied if $\mathcal{A}$ is input-complete, for $\mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\mathcal{I}$ **tioco** $\mathcal{A}$ is equivalent to $\mathcal{I} \preceq \mathcal{A}$. By transitivity of $\preceq$, it follows that io-refinement preserves conformance (see Proposition 4.1).

Lemma 4.1. *The io-refinement $\preceq$ is a preorder relation.*

Proof. The relation $\preceq$ is trivially reflexive and we prove that it is transitive.

Suppose that $\mathcal{A} \preceq \mathcal{B}$ and $\mathcal{B} \preceq \mathcal{C}$. By definition of $\preceq$ we have:

$$\forall \sigma \in \mathsf{Traces}(\mathcal{B}),\ out(\mathcal{A}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{B}\ \texttt{after}\ \sigma) \quad (1)$$
$$\forall \sigma \in \mathsf{Traces}(\mathcal{A}),\ in(\mathcal{B}\ \texttt{after}\ \sigma) \subseteq in(\mathcal{A}\ \texttt{after}\ \sigma) \quad (2) \quad \textit{and}$$
$$\forall \sigma \in \mathsf{Traces}(\mathcal{C}),\ out(\mathcal{B}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{C}\ \texttt{after}\ \sigma) \quad (3)$$
$$\forall \sigma \in \mathsf{Traces}(\mathcal{B}),\ in(\mathcal{C}\ \texttt{after}\ \sigma) \subseteq in(\mathcal{B}\ \texttt{after}\ \sigma) \quad (4)$$

We want to prove that $\mathcal{A} \preceq \mathcal{C}$ thus that

$$\forall \sigma \in \mathsf{Traces}(\mathcal{C}),\ out(\mathcal{A}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{C}\ \texttt{after}\ \sigma) \quad (5)$$
$$\forall \sigma \in \mathsf{Traces}(\mathcal{A}),\ in(\mathcal{C}\ \texttt{after}\ \sigma) \subseteq in(\mathcal{A}\ \texttt{after}\ \sigma) \quad (6)$$

In order to prove (5), let $\sigma \in \mathsf{Traces}(\mathcal{C})$, and examine the two cases:

- If $\sigma \in \mathsf{Traces}(\mathcal{B}) \cap \mathsf{Traces}(\mathcal{C})$ then (1) and (3) imply $out(\mathcal{A}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{B}\ \texttt{after}\ \sigma)$ and $out(\mathcal{B}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{C}\ \texttt{after}\ \sigma)$. Thus $out(\mathcal{A}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{C}\ \texttt{after}\ \sigma)$ and we are done.
- If $\sigma \in \mathsf{Traces}(\mathcal{C}) \setminus \mathsf{Traces}(\mathcal{B})$, there exist $\sigma', \sigma'' \in (\Sigma_{obs} \sqcup \mathbb{R}_{\geq 0})^*$ and $a \in \Sigma_{obs} \sqcup \mathbb{R}_{\geq 0}$ such that $\sigma = \sigma'.a.\sigma''$ with $\sigma' \in \mathsf{Traces}(\mathcal{B}) \cap \mathsf{Traces}(\mathcal{C})$ and $\sigma'.a \in \mathsf{Traces}(\mathcal{C}) \setminus \mathsf{Traces}(\mathcal{B})$. As $\mathcal{B} \preceq \mathcal{C}$, by (4) we get that $a \in \Sigma_! \sqcup \mathbb{R}_{\geq 0}$. But as $\mathcal{A} \preceq \mathcal{B}$, and $\sigma' \in \mathsf{Traces}(\mathcal{B})$, the condition (1) induces that $out(\mathcal{A}\ \texttt{after}\ \sigma') \subseteq out(\mathcal{B}\ \texttt{after}\ \sigma')$, and then $\sigma'.a \in \mathsf{Traces}(\mathcal{C}) \setminus \mathsf{Traces}(\mathcal{A})$. We deduce that $out(\mathcal{A}\ \texttt{after}\ \sigma'.a) = \emptyset$ and thus $out(\mathcal{A}\ \texttt{after}\ \sigma) = \emptyset \subseteq out(\mathcal{C}\ \texttt{after}\ \sigma)$.

The proof of (6) is similar. □

Proposition 4.1. *If $\mathcal{A} \preceq \mathcal{B}$ then $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A})$ $(= \mathcal{I}(\mathcal{B}))$, $\mathcal{I}$ **tioco** $\mathcal{A} \Rightarrow \mathcal{I}$ **tioco** $\mathcal{B}$.*

Proof. This proposition is a direct consequence of the transitivity of $\preceq$. In fact when $\mathcal{I}$ is input-complete, by definition $\forall \sigma \in \mathsf{Traces}(\mathcal{I}), in(\mathcal{I}\ \texttt{after}\ \sigma) = \Sigma_?$, thus condition (ii) of $\preceq$ trivially holds: $\forall \sigma \in \mathsf{Traces}(\mathcal{I}), in(\mathcal{A}\ \texttt{after}\ \sigma) \subseteq in(\mathcal{I}\ \texttt{after}\ \sigma)$. Thus $\mathcal{I}$ **tioco** $\mathcal{A}$ (which is defined by $\forall \sigma \in \mathsf{Traces}(\mathcal{A}), out(\mathcal{I}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{A}\ \texttt{after}\ \sigma)$) is equivalent to $\mathcal{I} \preceq \mathcal{A}$. Now suppose $\mathcal{A} \preceq B$ and $\mathcal{I}$ **tioco** $\mathcal{A}$ then the transitivity of $\preceq$ gives $\mathcal{I}$ **tioco** $\mathcal{B}$. □

Remark 4.1. *Unfortunately, the converse of Proposition 4.1 is in general false, already in the untimed case. This is illustrated in Figure 4.6. It is clear that the automaton $\mathcal{A}$ accepts all implementations. $\mathcal{B}$ also accepts all implementations as, from the conformance point of view, when a specification does not specify an input after a trace, this is equivalent to specifying this input and then to accept the universal language on Σ_{obs}. Thus $\mathcal{I}$ **tioco** $\mathcal{A} \Rightarrow \mathcal{I}$ **tioco** B. However $\neg(\mathcal{A} \preceq \mathcal{B})$ as $in(\mathcal{B}\ \texttt{after}\ \epsilon) = \{a\}$ but $in(\mathcal{A}\ \texttt{after}\ \epsilon) = \emptyset$. Notice that this example also works for the untimed case in the* **ioco** *conformance theory.*



Figure 4.6: Counter-example to converse of Proposition 4.1.

As a corollary of Proposition 4.1, we get that io-refinement preserves soundness of test suites:

Corollary 4.1. *If $A \preceq B$ then any sound test suite for $\mathcal{B}$ is also sound for $\mathcal{A}$.*

Proof. Let $\mathcal{TS}$ be a sound test suite for $\mathcal{B}$. By definition, for any $\mathcal{I} \in \mathcal{I}(\mathcal{B})$, for any $\mathcal{TC} \in \mathcal{TS}$, $\mathcal{I}$ `fails` $\mathcal{TC} \Rightarrow \neg(\mathcal{I}$ **tioco** $\mathcal{B})$. As we have $\mathcal{A} \preceq \mathcal{B}$, by Proposition 4.1, we obtain $\neg(\mathcal{I}$ **tioco** $\mathcal{B}) \Rightarrow \neg(\mathcal{I}$ **tioco** $\mathcal{A})$ which implies that for any $\mathcal{I} \in \mathcal{I}(\mathcal{B})$, for any $\mathcal{TC} \in \mathcal{TS}, \mathcal{I}$ `fails` $\mathcal{TC} \Rightarrow \neg(\mathcal{I}$ **tioco** $\mathcal{A})$. Thus $\mathcal{TS}$ is also sound for $\mathcal{A}$. □

In the sequel, this corollary will justify our methodology: from $\mathcal{A}$ a non-deterministic TAIO, build a deterministic io-abstraction $\mathcal{B}$ of $\mathcal{A}$, then any test case generated from $\mathcal{B}$ and sound is also sound for $\mathcal{A}$.

4.3 Off-line test case generation

In this section, we describe the off-line generation of test cases from timed automata specifications and test purposes. We first define test purposes, their role in test generation and their formalization as OTAIOs. We then detail the process of off-line test selection guided by test purposes, which uses the approximate determinization just defined. We also prove properties of generated test cases with respect to conformance and test purposes.

4.3.1 Test purposes

In testing practice, especially when test cases are generated manually, each test case has a particular objective, informally described by a sentence called test purpose. In formal test generation, test purposes should be formal models interpreted as means to select behaviors to be tested, either focusing on usual behaviors, or on suspected errors in implementations [JJ05], thus typically reachability properties. They complement other selection mechanisms such as coverage methods [ZHM97] which, contrary to test purposes, are most often based on syntactical criteria rather than semantic aspects. Moreover, the set of goals covering a given criterion (*e.g.* states, transitions, etc) may be translated into a set of test purposes, each test purpose focusing on one such goal.

As test purposes are selectors of behaviors, a natural way to formalize them is to use a logical formula characterizing a set of behaviors or an automaton accepting those behaviors. In this work we choose to describe test purposes as OTAIOs equipped with accepting states. The motivation is to use a model close to the specification model, easing the description of targeted specification behaviors. The following definition formalizes test purposes, and some alternatives are discussed in Section 4.4.

Definition 4.9 (Test purpose)**.** *Let $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{A}}, \emptyset, M^{\mathcal{A}}, \mathsf{Inv}^{\mathcal{A}}, E^{\mathcal{A}})$ be a TAIO specification. A* test purpose *for $\mathcal{A}$ is a pair $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$ where:*

- *$\mathcal{TP} = (L^{\mathcal{TP}}, \ell_0^{\mathcal{TP}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}}, M^{\mathcal{TP}}, \mathsf{Inv}^{\mathcal{TP}}, E^{\mathcal{TP}})$ is a complete OTAIO (in particular $\mathsf{Inv}^{\mathcal{TP}}(\ell) =$ `true` for any $\ell \in L^{\mathcal{TP}}$) with $X_o^{\mathcal{TP}} = X_p^{\mathcal{A}}$ ($\mathcal{TP}$ observes proper clocks of $\mathcal{A}$) and $X_p^{\mathcal{TP}} \cap X_p^{\mathcal{A}} = \emptyset$,*

- $\mathsf{Accept}^{\mathcal{TP}} \subseteq L^{\mathcal{TP}}$ *is a subset of trap locations.*

In the following, we will sometimes abuse notations and use $\mathcal{TP}$ instead of $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$. During the test generation process, test purposes are synchronized with the specification, and together with their Accept locations, they will play the role of acceptors of timed behaviors. They are non-intrusive in order not to constrain behaviors of the specification. This explains why they are complete, thus allowing all actions in all locations, and are not constrained by invariants. They observe behaviors of specifications by synchronizing with their actions (inputs, outputs and internal actions) and their proper clocks (by the definition of the product (Definition 4.3), observed clocks of $\mathcal{TP}$ are proper clocks of $\mathcal{A}$, which mean that $\mathcal{TP}$ does not reset those clocks). However, in order to add some flexibility in the description of timed behaviors, they may have their own proper clocks.

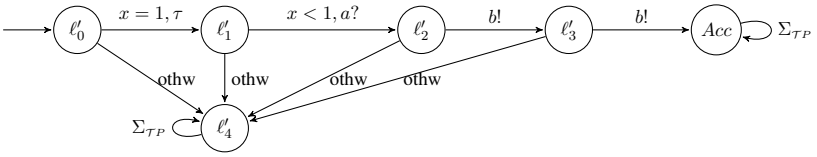


Figure 4.7: Test purpose $\mathcal{TP}$.

Running example Figure 4.7 represents a test purpose $\mathcal{TP}$ for the specification $\mathcal{A}$ of Figure 4.1. This one has no proper clock and observes the unique clock x of $\mathcal{A}$. It accepts sequences where τ occurs at $x = 1$, followed by an input a at $x < 1$ (thus focusing on the lower branch of $\mathcal{A}$ where x is reset), and two subsequent b's. The label $othw$ (for otherwise) on a transition is an abbreviation for the complement of specified transitions leaving the same location. For example in location ℓ'_1, $othw$ stands for $\{(\mathtt{true}, \tau), (\mathtt{true}, b!), (x \geq 1, a?)\}$.

4.3.2 Principle of test generation

Given a specification TAIO $\mathcal{A}$ and a test purpose $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$, the aim is to build a sound and, if possible strict test case $(\mathcal{TC}, \textbf{Verdicts})$ focusing on behaviors accepted by $\mathcal{TP}$. As $\mathcal{TP}$ accepts sequences of $\mathcal{A}$, but test cases observe timed traces, the intention is that $\mathcal{TC}$ should deliver **Pass** verdicts on traces of sequences of $\mathcal{A}$ accepted by $\mathcal{TP}$ in $\mathsf{Accept}^{\mathcal{TP}}$. This property is formalized by the following definition:

Definition 4.10. *A test suite $\mathcal{TS}$ for $\mathcal{A}$ and $\mathcal{TP}$ is said to be* precise *if for any test case $\mathcal{TC}$ in $\mathcal{TS}$, for any timed observation σ in $\mathsf{Traces}(\mathcal{TC})$, $\mathtt{Verdict}(\sigma, \mathcal{TC}) =$ **Pass** if and only if $\sigma \in \mathsf{Traces}(\mathsf{Seq}(\mathcal{A}\uparrow^{(X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}})}) \cap \mathsf{Seq}_{\mathsf{Accept}^{\mathcal{TP}}}(\mathcal{TP}))$.*

Let $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{A}}, \emptyset, M^{\mathcal{A}}, \mathsf{Inv}^{\mathcal{A}}, E^{\mathcal{A}})$ be the specification TAIO, and $\mathcal{TP} = (L^{\mathcal{TP}}, \ell_0^{\mathcal{TP}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}}, M^{\mathcal{TP}}, \mathsf{Inv}^{\mathcal{TP}}, E^{\mathcal{TP}})$ be a test purpose for $\mathcal{A}$, with its set $\mathsf{Accept}^{\mathcal{TP}}$ of accepting locations. The generation of a test case $\mathcal{TC}$ from $\mathcal{A}$ and $\mathcal{TP}$ proceeds in several steps. First, sequences of $\mathcal{A}$ accepted by $\mathcal{TP}$ are identified by the computation of the product $\mathcal{P}$ of those OTAIOs. Then a determinization step is necessary to characterize conformant traces as well as traces of accepted sequences. Then the resulting deterministic TAIO $\mathcal{DP}$ is transformed into a test case TAIO $\mathcal{TC}'$ with verdicts assigned to states. Finally, the test case $\mathcal{TC}$ is obtained by a selection step which tries to avoid some **Inconc** verdicts. The different steps of the test generation process from $\mathcal{A}$ and $\mathcal{TP}$ are detailed in the following paragraphs.

Computation of the product:

First, the product $\mathcal{P} = \mathcal{A} \times \mathcal{TP}$ is built (see Definition 4.3 for the definition of the product), associated with the set of marked locations $\mathsf{Accept}^{\mathcal{P}} = L^{\mathcal{A}} \times \mathsf{Accept}^{\mathcal{TP}}$. Let $P = (L^{\mathcal{P}}, \ell_0^{\mathcal{P}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{P}}, X_o^{\mathcal{P}}, M^{\mathcal{P}}, \mathsf{Inv}^{\mathcal{P}}, E^{\mathcal{P}})$. As $X_o^{\mathcal{TP}} = X_p^{\mathcal{A}}$, we get $X_o^{\mathcal{P}} = \emptyset$ and $X_p^{\mathcal{P}} = X_p^{\mathcal{A}} \sqcup X_p^{\mathcal{TP}}$, thus $\mathcal{P}$ is in fact a TAIO.

The effect of the product is to unfold $\mathcal{A}$ and to mark locations of the product by $\mathsf{Accept}^{\mathcal{P}}$, so that sequences of $\mathcal{A}$ accepted by $\mathcal{TP}$ are identified. As $\mathcal{TP}$ is complete, $\mathsf{Seq}(\mathcal{TP}) \downarrow_{X_p^{\mathcal{TP}}} = (\mathbb{R}_{\geq 0} \times (\Sigma^{\mathcal{TP}} \times 2^{X_o^{\mathcal{TP}}}))^*$, thus, by the properties of the product (see equation 4.2), $\mathsf{Seq}(\mathcal{P}) \downarrow_{X_p^{\mathcal{TP}}} = \mathsf{Seq}(\mathcal{A})$ *i.e.*the sequences of the product after removing resets of proper clocks of $\mathcal{TP}$ are the sequences of $\mathcal{A}$. As a consequence $\mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{A})$, which entails that $\mathcal{P}$ and $\mathcal{A}$ define the same sets of conformant implementations.

Considering accepted sequences of the product $\mathcal{P}$, by equation 4.3 we get the following equality $\mathsf{Seq}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}) = \mathsf{Seq}(\mathcal{A}\uparrow^{(X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}})}) \cap \mathsf{Seq}_{\mathsf{Accept}^{\mathcal{TP}}}(\mathcal{TP})$, which induces the desired characterization of accepted traces: $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}) = \mathsf{Traces}(\mathsf{Seq}(\mathcal{A}\uparrow^{(X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}})}) \cap \mathsf{Seq}_{\mathsf{Accept}^{\mathcal{TP}}}(\mathcal{TP}))$.

Writing $\mathsf{pref}(T)$ for the set of prefixes of traces in a set of traces T, we note $\mathsf{RTraces}(\mathcal{A}, \mathcal{TP}) = \mathsf{Traces}(\mathcal{A}) \setminus \mathsf{pref}(\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}))$ for the set of traces of $\mathcal{A}$ which are not prefixes of accepted traces of $\mathcal{P}$. In the sequel, the principle of test selection will be to try to select traces in $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P})$ (and assign to them the **Pass** verdict) and to try to avoid or at least detect (with an **Inconc** verdict) those traces in $\mathsf{RTraces}(\mathcal{A}, \mathcal{TP})$, as these traces cannot be prefixes of traces of sequences satisfying the test purpose.

Running example Figure 4.8 represents the product $\mathcal{P}$ for the specification $\mathcal{A}$ in Figure 4.1 and the test purpose $\mathcal{TP}$ in Figure 4.7. As $\mathcal{TP}$ describes one branch of $\mathcal{A}$, the product is very simple in this case, *e.g.* intersection of guards are trivial. The only difference with $\mathcal{A}$ is the tagging with $\mathsf{Accept}^{\mathcal{P}}$.

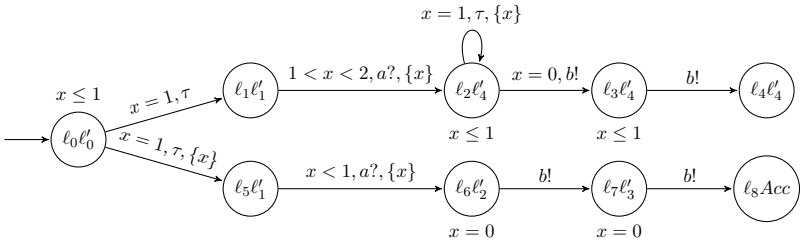


Figure 4.8: Product $\mathcal{P} = \mathcal{A} \times \mathcal{TP}$.

Approximate determinization of $\mathcal{P}$ into $\mathcal{DP}$:

We now want to transform $\mathcal{P}$ into a deterministic TAIO $\mathcal{DP}$ such that $\mathcal{P} \preceq \mathcal{DP}$, which by Proposition 4.1 will entail that implementations conformant to $\mathcal{P}$ (thus to $\mathcal{A}$) are still conformant to $\mathcal{DP}$. If $\mathcal{P}$ is already deterministic, we simply take $\mathcal{DP} = \mathcal{P}$. Otherwise, our game approach for the deteminization of timed automata is used (see the previous chapter for definitions and useful extensions). It is adapted to the context of testing for building a deterministic io-abstraction and it deals with invariants and internal actions. Indeed, as seen in Section 4.2, the io-refinement is simply the (Σ_1, Σ_2)-refinement where Σ_1 and Σ_2 are respectively the set of inputs and the set of outputs. The correctness of the determinization step is thus based on the following corollary of Proposition 3.4.

Corollary 4.1. *Let $\mathcal{A}$ be a TA, and $k, M^{\mathcal{B}} \in \mathbb{N}$. For any strategy Π of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M^{\mathcal{B}})}$, $\mathcal{B} = \mathsf{Aut}(\Pi)$ is a deterministic timed automaton over resources $(k, M^{\mathcal{B}})$ which io-abstracts $\mathcal{A}$. Moreover, if Π is winning, then $\mathsf{Traces}(\mathcal{A}) = \mathsf{Traces}(\mathcal{B})$.*

Then, the user fixes some resources $(k, M^{\mathcal{DP}})$, then a deterministic io-abstraction $\mathcal{DP}$ of $\mathcal{P}$ with resources $(k, M^{\mathcal{DP}})$ is computed thanks to our game approach. $\mathcal{DP}$ is equipped with the set of marked locations $\mathsf{Accept}^{\mathcal{DP}}$ consisting of locations in $L^{\mathcal{DP}}$ containing some configuration whose location is in $\mathsf{Accept}^{\mathcal{P}}$. As a consequence traces of $\mathcal{DP}$ which are traces of sequences accepted by $\mathcal{P}$ in $\mathsf{Accept}^{\mathcal{P}}$ are accepted by $\mathcal{DP}$ in $\mathsf{Accept}^{\mathcal{DP}}$, formally $\mathsf{Traces}(\mathcal{DP}) \cap \mathsf{Traces}(\mathsf{Seq}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P})) = \mathsf{Traces}(\mathcal{DP}) \cap \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}) \subseteq \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{DP}}}(\mathcal{DP})$. This means that extra accepted traces may be added due to over-approximations, some traces may be lost (including accepted ones) by under-approximations, but if the under-approximation preserves some traces that are accepted in $\mathcal{P}$, these are still accepted in $\mathcal{DP}$. If the determinization is exact (or $\mathcal{P}$ is already deterministic), of course we get more precise relations between the traces and accepted traces of $\mathcal{P}$ and $\mathcal{DP}$, namely $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$ and $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{DP}}}(\mathcal{DP}) = \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P})$.

Running example Figure 4.9 partially represents the game $\mathcal{G}_{\mathcal{P},(1,2)}$ for the TAIO $\mathcal{P}$ of Figure 4.8 where, for readability reasons, some behaviors not co-reachable from $\mathsf{Accept}^{\mathcal{DP}}$ (dotted green states) are omitted. A strategy Π for Determinizator is represented by bold arrows. Π is not winning (the unsafe configuration, in gray, is unavoidable from the initial state), and in fact an approximation is performed. $\mathcal{DP}$, represented in Figure 4.10 is simply obtained from $\mathcal{G}_{\mathcal{P},(1,2)}$ and the strategy Π by merging transitions of Spoiler and those of Determinizator in the strategy.

Generating $\mathcal{TC}'$ from $\mathcal{DP}$:

The next step consists in building a test case $(\mathcal{TC}', \textbf{Verdicts})$ from $\mathcal{DP}$. The main point is the computation of verdicts. **Pass** verdicts are simply defined from $\mathsf{Accept}^{\mathcal{DP}}$. **Fail** verdicts that should detect unexpected outputs and delays, rely on a complementation. The difficult part is the computation of **Inconc** states which should detect when $\mathsf{Accept}^{\mathcal{DP}}$ is not reachable (or equivalently **None** states, those states where $\mathsf{Accept}^{\mathcal{DP}}$ is still reachable) and thus relies on an analysis of the co-reachability to locations $\mathsf{Accept}^{\mathcal{DP}}$. Another interesting point is the treatment of invariants. First $\mathcal{TC}'$ will have no invariants (which ensures that it is non-blocking). Second, invariants in $\mathcal{DP}$ are shifted to guards in $\mathcal{TC}'$ and in the definition of **Fail** so that test cases check that the urgency specified in $\mathcal{A}$ is satisfied by $\mathcal{I}$.

The test case constructed from $\mathcal{DP} = (L^{\mathcal{DP}}, \ell_0^{\mathcal{DP}}, \Sigma_?^{\mathcal{DP}}, \Sigma_!^{\mathcal{DP}}, \emptyset, X_p^{\mathcal{DP}}, \emptyset, M^{\mathcal{DP}}, \mathsf{Inv}^{\mathcal{DP}}, E^{\mathcal{DP}})$ and $\mathsf{Accept}^{\mathcal{DP}}$ is the pair $(\mathcal{TC}', \textbf{Verdicts})$ where:

- $\mathcal{TC}' = (L^{\mathcal{TC}'}, \ell_0^{\mathcal{TC}'}, \Sigma_?^{\mathcal{TC}'}, \Sigma_!^{\mathcal{TC}'}, \emptyset, X_p^{\mathcal{TC}'}, \emptyset, M^{\mathcal{TC}'}, \mathsf{Inv}^{\mathcal{TC}'}, E^{\mathcal{TC}'})$ is the TAIO such that:
  - $L^{\mathcal{TC}'} = L^{\mathcal{DP}} \sqcup \{\ell_{\textbf{Fail}}\}$ where $\ell_{\textbf{Fail}}$ is a new location;
  - $\ell_0^{\mathcal{TC}'} = \ell_0^{\mathcal{DP}}$ is the initial location;
  - $\Sigma_?^{\mathcal{TC}'} = \Sigma_!^{\mathcal{DP}} = \Sigma_!^{\mathcal{A}}$ and $\Sigma_!^{\mathcal{TC}'} = \Sigma_?^{\mathcal{DP}} = \Sigma_?^{\mathcal{A}}$, *i.e.*input/output alphabets are mirrored in order to reflect the opposite role of actions in the synchronization of $\mathcal{TC}'$ and $\mathcal{I}$;
  - $X_p^{\mathcal{TC}'} = X_p^{\mathcal{DP}}$ and $X_o^{\mathcal{TC}'} = X_o^{\mathcal{DP}} = \emptyset$;
  - $M^{\mathcal{TC}'} = M^{\mathcal{DP}}$;
  - $\mathsf{Inv}^{\mathcal{TC}'}(\ell) = \mathtt{true}$ for any $\ell \in L^{\mathcal{TC}'}$;

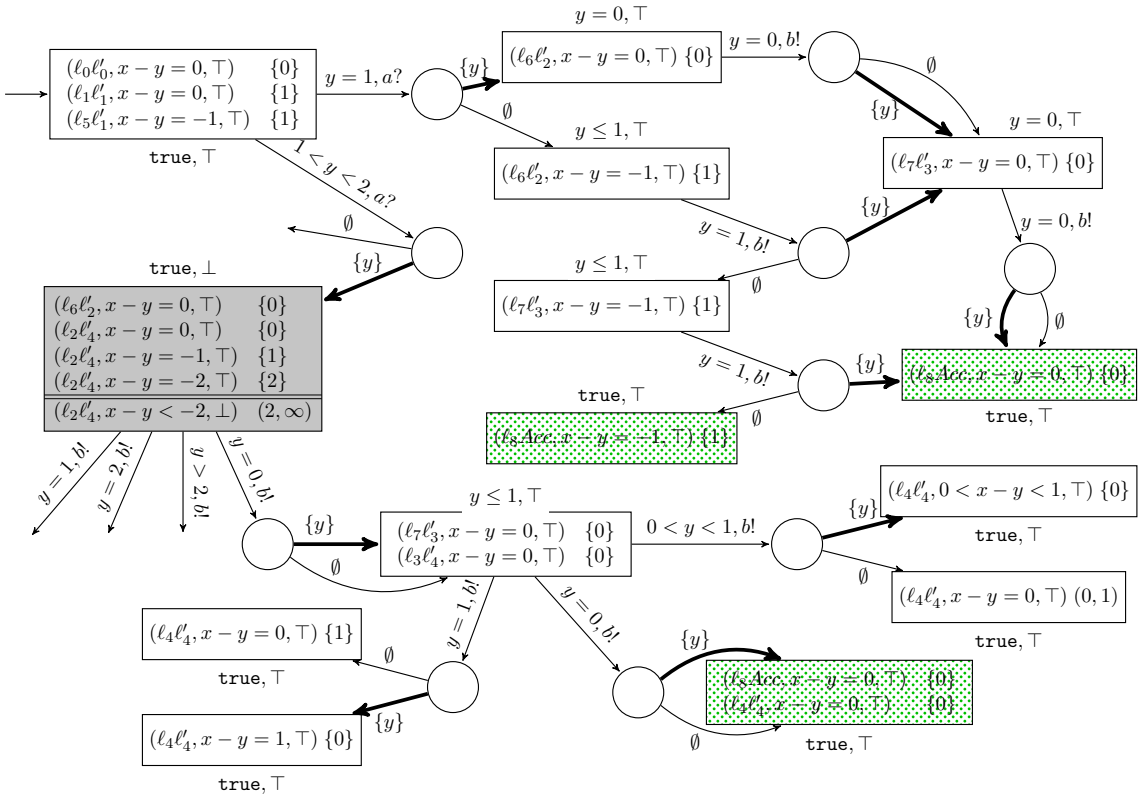


Figure 4.9: Game $\mathcal{G}_{\mathcal{P},(1,2)}$.

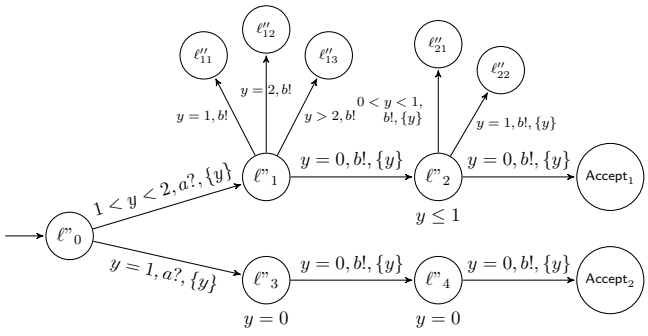


Figure 4.10: Deterministic automaton $\mathcal{DP} = Aut(\Pi)$.

– $E^{\mathcal{TC}'} = E^{\mathcal{DP}}_{\mathsf{Inv}} \sqcup E_{\ell_{\mathbf{Fail}}}$ where

$$E^{\mathcal{DP}}_{\mathsf{Inv}} = \{(\ell, g \wedge \mathsf{Inv}^{\mathcal{DP}}(\ell), a, X', \ell') \mid (\ell, g, a, X', \ell') \in E^{\mathcal{DP}}\} \text{ and}$$
$$E_{\ell_{\mathbf{Fail}}} = \left\{ (\ell, \bar{g} \wedge \mathsf{Inv}^{\mathcal{DP}}(\ell), a, X_p^{\mathcal{TC}'}, \ell_{\mathbf{Fail}}) \;\middle|\; \begin{array}{l} \ell \in L^{\mathcal{DP}},\ a \in \Sigma^{\mathcal{DP}}_! \\ \text{and } \bar{g} = \neg \bigvee_{(\ell,g,a,X',\ell') \in E^{\mathcal{DP}}} g \end{array} \right\}$$

- **Verdicts** is the partition of $S^{\mathcal{DP}}$ defined as follows:
  - **Pass** $= \bigcup_{\ell \in \mathsf{Accept}^{\mathcal{DP}}} (\{\ell\} \times \mathsf{Inv}^{\mathcal{DP}}(\ell))$,
  - **None** $=$ `coreach`$(\mathcal{DP}, \mathbf{Pass}) \setminus \mathbf{Pass}$,
  - **Fail** $= \{\ell_{\mathbf{Fail}}\} \times \mathbb{R}^{X^{\mathcal{TC}'}}_{\geq 0} \sqcup \{(\ell, \neg \mathsf{Inv}^{\mathcal{DP}}(\ell)) | \ell \in L^{\mathcal{DP}}\}$;
  - **Inconc** $= S^{\mathcal{DP}} \setminus (\mathbf{Pass} \sqcup \mathbf{Fail} \sqcup \mathbf{None})$,

The important points to understand in the construction of $\mathcal{TC}'$ are the completion to **Fail** and the computation of **None**, which, together with **Pass**, define **Inconc** by complementation.

For the completion to **Fail**, the idea is to detect unspecified outputs and delays with respect to $\mathcal{DP}$. Remember that outputs of $\mathcal{DP}$ are inputs of $\mathcal{TC}'$. Moreover, authorized delays in $\mathcal{DP}$ are defined by invariants, but remember that test cases have no invariants (they are `true` in all locations). First, all states in $(\ell, \neg \mathsf{Inv}^{\mathcal{DP}}(\ell)), \ell \in L^{\mathcal{DP}}$, *i.e.*states where the invariant runs out, are put into **Fail** which reflects the counterpart in $\mathcal{TC}'$ of the urgency in $\mathcal{DP}$. Then, in each location ℓ, the invariant $\mathsf{Inv}^{\mathcal{DP}}(\ell)$ in $\mathcal{DP}$ is removed and shifted to guards of all transitions leaving ℓ in $\mathcal{TC}'$, as defined in $E^{\mathcal{DP}}_I$. Second, in any location ℓ, for each input $a \in \Sigma^{\mathcal{TC}'}_? = \Sigma^{\mathcal{DP}}_!$, a transition leading to $\ell_{\mathbf{Fail}}$ is added, labeled with a, and whose guard is the conjunction of $\mathsf{Inv}(\ell)$ with the negation of the disjunction of all guards of transitions labeled by a and leaving ℓ (thus `true` if no a-action leaves ℓ), as defined in $E_{\ell_{\mathbf{Fail}}}$. It is then easy to see that $\mathcal{TC}'$ is input-complete in all states.

The computation of **None** is based on an analysis of the co-reachability to **Pass**. **None** contains all states co-reachable from locations in **Pass**. Notice that the set of states `coreach`$(\mathcal{DP}, \mathbf{Pass})$, and thus **None**, can be computed symbolically as usual in the region graph of $\mathcal{DP}$, or more efficiently using zones.

Running example Figure 4.11 represents the test case $\mathcal{TC}'$ obtained from $\mathcal{DP}$. For readability reasons, we did not represent transitions in $E_{\ell_{\mathbf{Fail}}}$, except the one leaving $\ell"_0$. In fact these are removed in the next selection phase as they are only fireable from states where a verdict has already been issued. The rectangles attached to locations represent the verdicts in these locations when clock y progresses between 0 and 2, and after 2: dotted green for $Pass$, black for **None**, blue grid for **Inconc** and crosshatched red for **Fail**. For example, in $\ell"_2$, the verdict is initially **None**, becomes **Inconc** if no b is received immediately, and even **Fail** if no b is received before one time unit. Notice that in order to reach a **Pass** verdict, one should initially send a after one and strictly before two time units, and expect to receive two consecutive b's immediately after.

Selection of $\mathcal{TC}$:

So far, the construction of $\mathcal{TC}'$ determines **Verdicts**, but does not perform any selection of behaviors. A last step consists in trying to control the behavior of $\mathcal{TC}'$ in order to avoid **Inconc** states (thus stay in $\mathsf{pref}(\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}))$), because reaching **Inconc** means that **Pass** is unreachable, thus $\mathcal{TP}$ cannot be satisfied anymore. To this aim, guards of transitions of $\mathcal{TC}'$ are refined in the final test case $\mathcal{TC}$ in two complementary ways. First, transitions leaving a verdict state (**Fail**, **Inconc** or **Pass**) are useless,

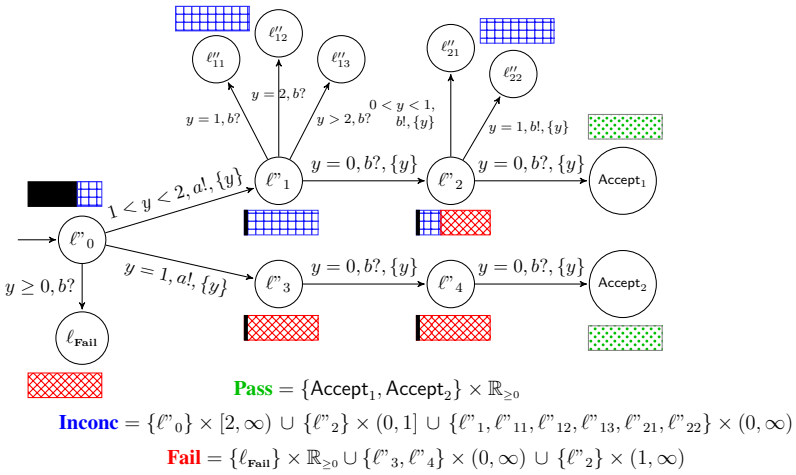


Figure 4.11: Test case $\mathcal{TC}'$ with verdicts

because the test case execution stops when a verdict is issued. Thus for each transition, the guard is intersected with the predicate characterizing the set of valuations associated with **None** in the source location. This does not change the verdict of traces. Second, transitions arriving in **Inconc** states and carrying outputs can be avoided (outputs are controlled by the test case), thus for any transition labeled by an output, the guard is intersected with the predicate characterizing **None** and **Pass** states in the target location (*i.e.* states that are not in **Inconc**, as **Fail** cannot be reached by an output). The effect is to suppress some traces leading to **Inconc** states. All in all, traces in $\mathcal{TC}$ are exactly those of $\mathcal{TC}'$ that traverse only **None** states (except for the last state), and do not end in **Inconc** with an output. This selection does not impact on the properties of test suites (soundness, strictness, precision and exhaustiveness) as will be seen later.

Running example Figure 4.12 represents the test case obtained after this selection phase. One can notice that locations $\ell''_{11}, \ell''_{12}, \ell''_{13}$ and ℓ''_{21}, ℓ''_{22} have been removed since they can only be reached from **Inconc** states, thus a verdict will have been emitted before reaching those locations. The avoidance of **Inconc** verdicts by outputs cannot be observed on this example. However, with a small modification of $\mathcal{A}$ consisting in adding initially the reception of an a before one time unit, and not followed by two b's but *e.g.* one c, the resulting transition labeled with $(0 \leq y < 1, a!)$ in $\mathcal{TC}'$ could be cut, producing the same $\mathcal{TC}$.

Remark 4.1. *Notice that in the example, falling into* **Inconc** *in* ℓ''_0 *could be avoided by adding the invariant* $y < 2$*, with the effect of forcing to output* a*. More generally, invariants can be added to locations by rendering outputs urgent in order to avoid* **Inconc***, while taking care of keeping test cases non-blocking,* i.e. *by ensuring that an output can be done just before the invariant becomes false. More precisely,* $I(\ell)$ *is the projection of* **None** *on* ℓ *if* $Inconc$ *is reachable by letting time elapse and it preserves the non-blocking property,* `true` *otherwise.*

Complexity

Let us discuss the complexity of the construction of $\mathcal{TC}$ from $\mathcal{DP}$. Note that the size of TAIO $\mathcal{TC}$ is linear in the size of $\mathcal{DP}$ but the difficulty lies in the computation of **Verdicts**. Computing

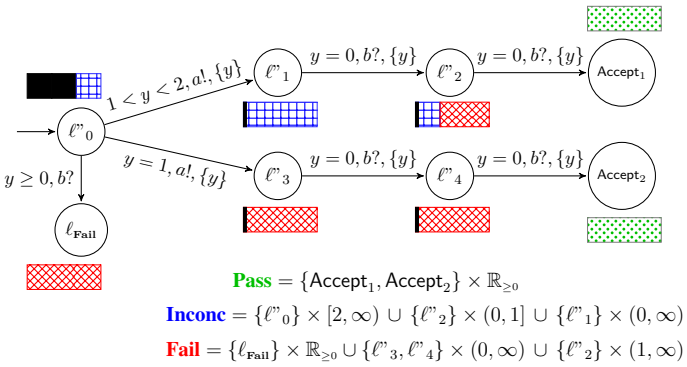


Figure 4.12: Final test case $\mathcal{TC}$ after selection

Pass is immediate. The set `coreach`(**Pass**) can be computed in polynomial time (more precisely in $\mathcal{O}(|L^{\mathcal{DP}}|.|X^{\mathcal{DP}}|.|M^{\mathcal{DP}}|)$). To explain this, observe that guards in the TAIO $\mathcal{DP}$ are regions and with each location ℓ is associated an initial region r_ℓ such that guards of transitions leaving ℓ are time successors of r_ℓ. Thus during the computation of `coreach`(**Pass**), for each location ℓ, one only needs to consider these $\mathcal{O}(|X^{\mathcal{DP}}|.|M^{\mathcal{DP}}|)$ different regions in order to determine the latest time-successor $r_\ell^{\max}$ of r_ℓ which is co-reachable from **Pass**. Then **None** states with location ℓ are exactly those within regions that are time-predecessors of $r_\ell^{\max}$. For the same reason (number of possible guards outgoing a given location) $E_{\ell_{\mathbf{Fail}}}$ can be computed in polynomial time. Last the **Fail** verdicts in locations (except for $\ell_{\mathbf{Fail}}$) are computed in linear time by complementing the invariants in $\mathcal{DP}$. The test selection can be done by inspecting all transitions: a transition is removed if either the source state is a verdict state, or it corresponds to an output action and the successor are **Inconc** states. This last step thus only requires linear time. To conclude, the overall complexity of construction of $\mathcal{TC}$ from $\mathcal{DP}$ is polynomial.

4.3.3 Test suite properties

We have presented the different steps for the generation of a TAIO test case from a TAIO specification and an OTAIO test purpose. The following results express their properties.

Theorem 4.1. *Any test case $\mathcal{TC}$ built by the procedure is sound for $\mathcal{A}$. Moreover, if $\mathcal{DP}$ is an exact approximation of $\mathcal{P}$ (*i.e.$\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$*), the test case $\mathcal{TC}$ is also strict and precise for $\mathcal{A}$ and $\mathcal{TP}$.*

The proof is detailed below, but we first give some intuition. As a preamble, notice that, as explained in the paragraph on test selection, traces of $\mathcal{TC}'$ are not affected by the construction of $\mathcal{TC}$. In particular, the transitions considered in the proof are identical in $\mathcal{TC}$ and $\mathcal{TC}'$. Soundness comes from the construction of $E_{\ell_{\mathbf{Fail}}}$ in $\mathcal{TC}$ and preservation of soundness by the approximate determinization $\mathcal{DP}$ of $\mathcal{P}$ given by Corollary 4.1. When $\mathcal{DP}$ is an exact determinization of $\mathcal{P}$, $\mathcal{DP}$ and $\mathcal{P}$ have same traces, which also equal traces of $\mathcal{A}$ since $\mathcal{TP}$ is complete. Strictness then comes from the fact that $\mathcal{DP}$ and $\mathcal{A}$ have the same non-conformant traces, which are captured by the definition of $E_{\ell_{\mathbf{Fail}}}$ in $\mathcal{TC}$. Precision comes from $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{DP}}}(\mathcal{DP}) = \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P})$ and from the definition of **Pass**.

When $\mathcal{DP}$ is not exact however, there is a risk that some behaviors allowed in $\mathcal{DP}$ are not in $\mathcal{P}$, thus some non-conformant behaviors are not detected, even if they are executed by $\mathcal{TC}$. Similarly, some **Pass** verdicts may be produced for non-accepted or even non-conformant behaviors. However,

if a trace in $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P})$ is present in $\mathcal{TC}$ and observed during testing, a **Pass** verdict will be delivered. In other words, precision is not always satisfied, but the "only if" direction of precision (Definition 4.10) is satisfied.

Proof. **Soundness:** To prove soundness, we need to show that for any $\mathcal{I} \in \mathcal{IO}(\mathcal{A})$, $\mathcal{I}\,\texttt{fails}\,\mathcal{TC}$ implies $\neg(\mathcal{I}\,\mathbf{tioco}\,\mathcal{A})$.

Assuming that $\mathcal{I}\,\texttt{fails}\,\mathcal{TC}$, there exists a trace $\sigma \in \mathsf{Traces}(\mathcal{I}) \cap \mathsf{Traces}_{\mathbf{Fail}}(\mathcal{TC})$. By the construction of the set **Fail** in $\mathcal{TC}$, there are two cases: either σ leads to a location $(\ell, \neg(\mathsf{Inv}(\ell))$ in $\mathcal{DP}$, or σ leads to a state with location $\ell_{\mathbf{Fail}}$. In the first case, $\sigma = \sigma'.\delta$ where $\sigma' \in \mathsf{Traces}(\mathcal{DP})$ and $\delta > 0$ violates the invariant in the location of $\mathcal{DP}$ $\texttt{after}$ σ', and in the second case, by the construction of $E_{\ell_{\mathbf{Fail}}}$, $\sigma = \sigma'.a$ where $\sigma' \in \mathsf{Traces}(\mathcal{DP})$ and $a \in \Sigma_!^{\mathcal{DP}}$ is unspecified in $\mathcal{DP}$ $\texttt{after}$ σ'. In both cases, by definition, this means that $\neg(\mathcal{I}\,\mathbf{tioco}\,\mathcal{DP})$, which proves that $\mathcal{TC}$ is sound for $\mathcal{DP}$. Now, as $\mathcal{DP}$ is an io-abstraction of $\mathcal{P}$ (*i.e.* $\mathcal{P} \preceq \mathcal{DP}$), by Corollary 4.1 this entails that $\mathcal{TC}$ is sound for $\mathcal{P}$. Finally, we have $\mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{A})$, which trivially implies that $\mathcal{A} \preceq \mathcal{P}$, and thus that $\mathcal{TC}$ is also sound for $\mathcal{A}$.

Strictness: For strictness, in the case where $\mathcal{DP}$ is an exact approximation of $\mathcal{P}$, we need to prove that for any $\mathcal{I} \in \mathcal{IO}(\mathcal{A})$, $\neg(\mathcal{I} \| \mathcal{TC}\,\mathbf{tioco}\,\mathcal{A})$ implies that $\mathcal{I}\,\texttt{fails}\,\mathcal{TC}$. Suppose that $\neg(\mathcal{I} \| \mathcal{TC}\,\mathbf{tioco}\,\mathcal{A})$. By definition, there exists a trace $\sigma \in \mathsf{Traces}(\mathcal{A})$ and $a \in out(\mathcal{I} \| \mathcal{TC}\ \texttt{after}\ \sigma)$ such that $a \notin out(\mathcal{A}\ \texttt{after}\ \sigma)$. Since $\mathcal{DP}$ is an exact approximation of $\mathcal{P}$, we have the equalities $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{A})$, thus $\sigma \in \mathsf{Traces}(\mathcal{DP})$ and $a \notin out(\mathcal{DP}\ \texttt{after}\ \sigma)$. By construction of **Fail** in $\mathcal{TC}$, it follows that $\sigma.a \in \mathsf{Traces}_{\mathbf{Fail}}(\mathcal{TC})$ which, together with $\sigma.a \in \mathsf{Traces}(\mathcal{I})$, implies that $\mathcal{I}\,\texttt{fails}\,\mathcal{TC}$. Thus $\mathcal{TC}$ is strict.

Precision: To prove precision, in the case of exact determinization, we have to show that for any trace σ, $\texttt{Verdict}(\sigma, \mathcal{TC}) = \mathbf{Pass} \iff \sigma \in \mathsf{Traces}(\mathsf{Seq}_{\mathsf{Accept}^{\mathcal{TP}}}(\mathcal{TP}) \cap \mathsf{Seq}(\mathcal{A}))$. The definition of $\mathbf{Pass} = \bigcup_{\ell \in \mathsf{Accept}^{\mathcal{DP}}} (\{\ell\} \times \mathsf{Inv}^{\mathcal{DP}}(\ell))$ in $\mathcal{TC}$ implies that a **Pass** verdict is produced for σ exactly when $\sigma \in \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{DP}}}(\mathcal{DP})$ which equals $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}) = \mathsf{Traces}(\mathsf{Seq}_{\mathsf{Accept}^{\mathcal{TP}}}(\mathcal{TP}) \cap \mathsf{Seq}(\mathcal{A}))$ when $\mathcal{DP}$ is exact. □

Running example The test case $\mathcal{TC}$ of Figure 4.12 comes from an approximate determinization. However, the approximation comes after reaching **Inconc** states. More precisely, in the gray state of the game in Figure 4.9, the approximation starts in the time interval $(2, \infty)$. This state corresponds to location $\ell"_1$ in $\mathcal{TC}$ where the verdict is **Inconc** as soon as a non null delay is observed. The test case is thus strict and precise, despite the over-approximation in the determinization phase.

In the following, we prove an exhaustiveness property of our test generation method when determinization is exact. For technical reasons, we need to restrict to a sub-class of TAIOs defined below. We discuss this restriction later.

Definition 4.11. *We say that an OTAIO $\mathcal{A}$ is* repeatedly observable *if from any state of $\mathcal{A}$, there is a future observable transition,* i.e. *$\forall s \in S^{\mathcal{A}}$, there exists μ such that $s \xrightarrow{\mu}$ and $Trace(\mu) \notin \mathbb{R}_{\geq 0}$.*

Theorem 4.2 (Exhaustiveness)**.** *Let $\mathcal{A}$ be a repeatedly observable TAIO which can be exactly determinized by our approach. Then the set of test cases that can be generated from $\mathcal{A}$ by our method is exhaustive.*

Proof. Let $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{A}}, \emptyset, M^{\mathcal{A}}, \mathsf{Inv}^{\mathcal{A}}, E^{\mathcal{A}})$ be the TAIO specification, and $\mathcal{I} = (L^{\mathcal{I}}, \ell_0^{\mathcal{I}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{I}}, X_p^{\mathcal{I}}, \emptyset, M^{\mathcal{I}}, \mathsf{Inv}^{\mathcal{I}}, E^{\mathcal{I}})$ any non-conformant implementation in $\mathcal{IO}(\mathcal{A})$. The idea is

now to prove that from $\mathcal{A}$ and $\mathcal{I}$, one can build a test purpose $\mathcal{TP}$ such that the test case $\mathcal{TC}$ built from $\mathcal{A}$ and $\mathcal{TP}$ may detect this non-conformance, *i.e.* $\mathcal{I}$ `fails` $\mathcal{TC}$.

By definition of $\neg(\mathcal{I}$ **tioco** $\mathcal{A})$, there exists $\sigma \in \mathsf{Traces}(\mathcal{A})$ and $a \in \Sigma_!^{\mathcal{A}} \sqcup \mathbb{R}_{\geq 0}$ such that $a \in out(\mathcal{I}$ `after` $\sigma)$ but $a \notin out(\mathcal{A}$ `after` $\sigma)$. Since $\mathcal{A}$ is repeatedly observable, there also exists $\delta \in \mathbb{R}_{\geq 0}$ and $b \in \Sigma_{obs}^{\mathcal{A}}$ such that $\sigma.\delta.b \in \mathsf{Traces}(\mathcal{A})$.

As $\mathcal{A}$ can be determinized exactly by our approach, there must exist some resources (k, M) and a strategy Π for Determinizator in the game $\mathcal{G}_{\mathcal{A},(k,M)}$ such that $\mathsf{Traces}(\mathsf{Aut}(\Pi)) = \mathsf{Traces}(\mathcal{A})$.

From the non-conformant implementation $\mathcal{I}$, a test purpose $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$ can be built, with $\mathcal{TP} = (L^{\mathcal{TP}}, \ell_0^{\mathcal{TP}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}}, M^{\mathcal{TP}}, \mathsf{Inv}^{\mathcal{TP}}, E^{\mathcal{TP}})$, $X_p^{\mathcal{TP}} = X_p^{\mathcal{I}} \sqcup X^{\mathsf{Aut}(\Pi)}$ and $X_o^{\mathcal{TP}} = \emptyset$, and $\sigma.\delta.b \in \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{TP}}}$ but none of its prefixes is in $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{TP}}}$. The construction of $\mathcal{TP}$ relies on the region graph of $\mathcal{I} \| \mathsf{Aut}(\Pi)$. First a TAIO $\mathcal{TP}'$ is built which recognizes exactly the traces read along the path corresponding to σ in the region graph of $\mathcal{I} \| \mathsf{Aut}(\Pi)$, followed by a transition b with the guard corresponding to the one in $\mathsf{Aut}(\Pi)$. In particular it recognizes the trace $\sigma.\delta.b$. The test purpose $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$ is then built such that $\mathcal{TP}$ accepts in its states $\mathsf{Accept}^{\mathcal{TP}}$ the traces of $\mathcal{TP}'$. Note that $\mathcal{TP}$ should be complete for Σ, thus locations of $\mathcal{TP}'$ should be completed by adding loops without resets for all actions in Σ_τ, and adding, for all observable actions, transitions to a trap location guarded with negations of their guards in $\mathcal{TP}'$.

Now consider our test generation method applied to $\mathcal{TP}$ and $\mathcal{A}$. First $\mathcal{P} = \mathcal{A} \times \mathcal{TP}$ is built, and we consider the game $\mathcal{G}_{\mathcal{A},(k',M')}$ with $k' = k + |X_p^{\mathcal{TP}}|$ and $M' = \max(M, M^{\mathcal{TP}})$. One can then define a strategy Π' composed of the strategy Π for the k first clocks, and following the resets of $\mathcal{TP}$ (which is deterministic) for the other clocks corresponding to those in $X_p^{\mathcal{TP}}$. The construction of $(\mathcal{DP}, \mathsf{Accept}^{\mathcal{DP}})$ following the strategy Π' thus ensures that $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$ and $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{DP}}}(\mathcal{DP}) = \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P})$.

Finally, let $\mathcal{TC}$ be the test case built from $\mathcal{DP}$. Observe that $\mathcal{TC}$ `after` $\sigma.\delta.b \subseteq$ **Pass**, but $\mathcal{TC}$ `after` $\sigma.\delta \not\subseteq$ **Pass**. As a consequence, $\mathcal{TC}$ `after` $\sigma \subseteq$ **None**. Moreover we have $a \notin out(\mathcal{A}$ `after` $\sigma)$, hence $\sigma.a \in \mathsf{Traces}_{\mathbf{Fail}}(\mathcal{TC})$ and as $\sigma.a \in \mathsf{Traces}(\mathcal{I})$, we can conclude that $\mathcal{I}$ `fails` $\mathcal{TC}$. □

Discussion:

The hypothesis that $\mathcal{A}$ is repeatedly observable is in fact not restrictive for a TAIO that is determinizable by our approach. Indeed, such a TAIO can be transformed into a repeatedly observable one with same conformant implementations, by first determinizing it, and then completing it as follows. In all locations, a transition labeled by an input is added, which goes to a trap state looping for all outputs, and is guarded by the negation of the union of guards of transitions for this input in the deterministic automaton.

When $\mathcal{A}$ cannot be determinized exactly, the risk is that some non-conformance may be undetectable. However, the theorem can be generalized to non-determinizable automata with no resets on internal action. Indeed, in this case, in the game with resources (k, M), where k is the length of the finite non-conformant trace $\sigma.a$, the strategy consisting in resetting a new clock at each observable action allows to remain exact until the observation of non-conformance (see remark after Theorem 4.1). The proof of theorem 4.2 can be adapted using this strategy.

4.4 Discussion and related work

Alternative definitions of test purposes

The definition of test purposes depends on the semantic level at which behaviors to be tested are described (*e.g.* sequences, traces). This induces a trade-off between the precision of the description of behaviors, and the cost of producing test suites. In this work, test purposes recognize timed sequences of the specification $\mathcal{A}$, by a synchronization with actions and observed clocks. They also have their own proper clocks for additional precision. The advantage is a fine tuning of selection. The price to be paid is that, for each test purpose, the whole sequence of operations, including determinization which may be costly, must be done. An alternative is to define test purposes recognizing timed traces rather than timed sequences. In this case, selection should be performed on a deterministic io-abstraction $\mathcal{B}$ of $\mathcal{A}$ obtained by an approximate determinization of $\mathcal{A}$. Then, test purposes should not refer to $\mathcal{A}$'s clocks as these are lost by the approximate determinization. Test purposes should then either observe $\mathcal{B}$'s clocks, and thus be defined after determinization, or use only proper clocks in order not to depend on $\mathcal{B}$, at the price of further restricting the expressive power of test purposes. In both cases, test purposes should preferably be deterministic in order to avoid a supplementary determinization after the product with $\mathcal{B}$. The main advantage of these approaches is that the specification is determinized only once, which reduces the cost of producing a test suite. However, the expressive power of test purposes is reduced.

Test execution

Once test cases are selected, it remains to execute them on a real implementation. As a test case is a TAIO, and not a simple timed trace, a number of decisions still need to be taken at each state of the test case: (1) whether to wait for a certain delay, or to receive an input or to send an output (2) which output to send, in case there is a choice. It is clear that different choices may lead to different behaviors and verdicts. Some of these choices can be made either randomly (*e.g.* choosing a random time delay, choosing between outputs, etc), or can be pre-established according to user-defined strategies. One such policy is to apply a technique similar to the control approach of [DLLN09] whose goal is to avoid $\mathsf{RTraces}(\mathcal{A}, \mathcal{TP})$.

Moreover, the tester's time observation capabilities are limited in practice: testers only dispose of a finite-precision digital clock (a counter) and cannot distinguish among observations which elude their clock precision. Our framework may take this limitation into account. In [KT09] assumptions on the tester's digital clock are explicitly modeled as a special TAIO called $Tick$, synchronized with the specification before test generation, then relying to the untimed case. We could imagine to use such a $Tick$ automaton differently, by synchronizing it with the resulting test case after generation.

Related work

As mentioned in the introduction, off-line test selection is in general restricted to deterministic automata or known classes of determinizable timed automata. An exception is the work of [KT09] which relies on an over-approximate determinization. Compared to this work, our approximate determinization is more precise (it is exact in more cases), it copes with outputs and inputs using over- and under-approximations, and preserves urgency in test cases as much as possible. Another exception is the work of [DLLN09], where the authors propose a game approach whose effect can be understood as a way to completely avoid $\mathsf{RTraces}(\mathcal{A}, \mathcal{TP})$, with the possible risk of missing some or even

all traces in $\mathsf{pref}(\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}))$. Our selection, which allows to lose this game and produce an **Inconc** verdict when this happens, is both more liberal and closer to usual practice.

In several related works [KCL98, END03], test purposes are used for test case selection from TAIOs. In all these works, test purposes only have proper clocks, thus cannot observe clocks of the specification.

It should be noticed that selection by test purposes can be used for test selection with respect to coverage criteria [ZHM97]. Those coverage criteria define a set of elements (generally syntactic ones) to be covered (*e.g.* locations, transitions, branches, etc). Each element can then be translated into a test purpose, the produced test suite covering the given criteria.

Conclusion

In this chapter, we proposed an application of our game approach for the determinization of timed automata, which is a contribution in itself. We presented a complete formalization for the automatic off-line generation of test cases from non-deterministic timed automata with inputs and outputs. The model of TAIOs is general enough to take into account non-determinism, partial observation and urgency. One main contribution is the ability to tackle non-deterministic specifications, thanks to our game approach for the determinization. Another main contribution is the selection of test cases with expressive test purposes described as OTAIOs having the ability to precisely select behaviors to be tested, based on clocks and actions of the specification as well as proper clocks. Test cases are generated as TAIOs using a symbolic co-reachability analysis of the observable behaviors of the specification guided by the test purpose.

Conclusion

In this part, we presented a game approach for the determinization of timed automata. Timed automata are not determinizable in general, hence if we do not manage to build a deterministic equivalent, we construct an approximate determinization trying to minimize the approximation. We thus combine features of two existing approaches: a pseudo algorithm for the exact determinization [BBBB09] and an algorithm which yields a deterministic over-approximation [KT09]. The underlying problem of the determinization of timed automata is that resets can be different along two runs reading the same word. Then, our goal is to find a good reset policy for the clocks of the deterministic timed automaton in order to preserve all the clock information which is needed. To do so, we naturally proposed to build a game. Our approach deals with invariants and ε-transitions, and can moreover provide abstractions combining under- and over-approximations. We improved both existing approaches, determinizing strictly more timed automata, yielding finer approximations and dealing with richer models.

The determinization is fundamental in verification and in particular for model-based testing. We presented how to use our determinization technique to generate sound test cases from specifications given as non-deterministic timed automata. More precisely, the specification model that we consider, includes partial observability (modeled by ε-transitions) and urgency (modeled by invariants). We proposed a general formal setting. In particular, we introduced an extension of timed automata which are able to observe the clocks of other timed automata. This model allowed us to define fine test purposes to select sets of sequences (traces labeled with resets) instead of only sets of traces. Our approach for the determinization combining under- and over-approximation is then suitable to treat differently inputs and outputs. As a consequence, even if the determinization is approximate, the generated test cases are sound for the original specification.

Part IV

Frequencies in Timed Automata

Introduction

In this part, we are interested in refinements of the verification process, considering some quantities such as costs along runs. Moreover, the size of the part of the language which satisfies a property; or the probability for a run to satisfy a property, can be a more interesting information than an answer "no, the model does not satisfy the property". Recently, a huge effort has been made to add quantitative aspects in the verification of timed automata. Several quantitative notions have thus been introduced and studied in timed automata. In this part, we propose to study the proportion of time elapsed in accepting locations along infinite runs. This quantity is called the frequency of a run and could model, for example, the failure rate in the long run.

Quantities in timed automata The notions of volume and entropy of timed languages have been defined and several methods to compute them have been developed [AD09a, AD09b, AD10]. This allows either to quantify the growth rate of a timed language, considering longer and longer timed words read by a timed automaton, or read along a symbolic path of a timed automaton. The importance of a symbolic path can thus be weighted with respect to the other behaviors of the timed automaton. Another way to express the importance of a symbolic path is to use probabilities. A probabilistic semantics for timed automata has been introduced and studied in [BBB$^+$08, BBBM08]. The non-determinism over the enabled moves and over the delays is resolved, in a fair way, using probability distributions. Then, one can decide, in one-clock timed automata, whether the probability to satisfy an ω-regular property, satisfies a threshold condition for a rational threshold. For example, one can check properties of the form "the probability to visit infinitely often a given location, is greater than $\frac{1}{3}$". The approach is based on an algorithm which allows to compute the probability to satisfy the property if it is a rational number, and to approximate it up to a given precision otherwise. However the techniques do not extend to timed automata with several clocks, mainly because of some convergence phenomena between clocks, which appear when two clocks are allowed. Remark that there are other models which combine probabilities and timing aspects such as continuous probabilistic timed automata [KNSS02] or stochastic Petri nets [Mol82].

In a dual way, quantities can also be associated directly with runs. Thus, costs have naturally been put on both transitions and locations to extend the model of timed automata [ATP01, BFH$^+$01]. The cost of a delay d in a location whose associated cost is c is $d * c$. The cost for a run is then the sum of the costs of the delays added to the sum of the costs of the edges fired. Another model, combining two kinds of costs, called respectively costs and rewards, has been studied in [BBL08]. The runs are infinite and the value associated to each run, called ratio, is the limit inf of the ratio of the sum of costs over the sum of rewards. The three above-mentioned papers address the problem of optimal scheduling. More precisely, in each of these works, one looks for a run with the smallest cost or ratio.

Quantities on two dimensions Quantities can thus be considered in timed automata on two dimensions. They can be used as measures of the importance of some behaviors. For example, it may allow to identify some marginal phenomena in timed automata which could be neglected in the model-checking of a property. On the other hand, quantities can be directly assigned to runs. Then, it allows to define quantitative languages, either considering languages where each word has a value reflecting "how much" it belongs to the language, or by considering languages containing all the words whose value is, for example, greater than a threshold.

Quantitative untimed languages: mean-payoff condition In the untimed framework, quantitative languages generalize classical languages, by assigning a real number to each word. They can be defined thanks to a variety of models such as weighted automata [Sch61], lattice automata [KL07] or quantitative structures [CCH$^+$05]. We focus on weighted automata because this model is similar to an untimed version of the model of timed automata with frequencies. One can consider finite or infinite words and assign the result of functions such as the maximal or minimal weight along a run, the sum of weights, the limit sup, the limit inf, the mean payoff (or limit average) or a discounted sum. The properties of such languages are studied in [CDH08]. The different functions can model a lot of problems. For example, the maximal weight can represent a peak power consumption and the sum can model a quantity of energy. The discounted sum models the fact that the later a failure happens, the less important it is [dAHM03]. The mean-payoff function have widely been studied for games [EM79] where two players have to respectively minimize and maximize some means over the costs. It can model the failure rate in the long run putting weights 1 on failure states and 0 on the others. In this part we aim at proposing a timed version of such a model, hence we are particularly interested by mean-payoff automata on which there is a focus in [CDE$^+$10]. Mean-payoff automata are weighted automata that assign to each infinite run the long-run average of the transition weights. This proportion may not converge, which is why mean-payoff automata are said LimSupAvg- or LimInfAvg-automata depending on the cluster point which is chosen in case of non-convergence. The value associated with a word w is then the supremum of the values assigned to runs reading w. One can thus consider quantitative problems such as the existence of a word whose value is greater than a fixed threshold, which is a quantitative variant of the traditional emptiness problem. The main contribution of [CDE$^+$10] is to present a class of quantitative languages defined by a subclass of mean-payoff automata which is closed under max the quantitative analogue of union, min the quantitative analogue to intersection, sum and numerical complement (all values are multiplied by -1). This class, noted C here, is inductively defined as follows: deterministic (LimSupAvg- or LimInfAvg-) automata are in C and if $\mathcal{A}_1$ and $\mathcal{A}_2$ belong to C then the max, the min and the sum of $\mathcal{A}_1$ and $\mathcal{A}_2$ are in C.

Contribution: new semantics for infinite runs in timed automata The usual acceptance condition for infinite runs in timed automata is the Büchi semantics, where a run is accepted if it visits infinitely often accepting locations. In particular, a run visiting accepting locations extremely rarely and/or for a very small duration will be accepted. However, accepting locations could model either states of a system in which we avoid to stay, for example corresponding to inactivity of machines, or states in which staying is beneficial, for example corresponding to an optimal productivity. In such contexts, the Büchi semantics does not seem to be suitable and it appears important to consider the quantity of time elapsed in a subset of particular locations. Depending on the context, it would thus be interesting to weight the acceptance of a run, taking into account the frequency of accepting locations.

In this part, we propose to consider timed automata with frequencies. The intuition is that we assign to each infinite run the value computed as the proportion of time elapsed in accepting locations

along the run. If this proportion does not converge, we arbitrarily take the limit sup. The frequency is similar to the mean-payoff function in weighted automata, and in particular, naturally models the failure rate in the long run.

Thanks to this quantity, one can define a first semantics simply considering as accepted, those runs whose frequency is positive. This is the frequency-based acceptance condition which is the closest to the Büchi acceptance. Nevertheless, the relative expressivenesses of these semantics are not comparable. More generally, one can use frequency-based constraints, of the form "greater than a threshold", as acceptance conditions and define quantitative languages. The model investigated in [CDE$^+$10] can be seen as an untime version of ours. Difficulties to perform language operations come from weights, and time would not really change the problem. We prefer to focus on the study of the set of frequencies of the runs in a timed automaton with the aim to decide language problems. The emptiness problem, for instance, corresponds to the question "Does there exist a run whose frequency is greater than $\frac{1}{2}$?". The universality problem is much harder. We prove, by a reduction from the universality problem for finite words in timed automata, that it is non-primitive recursive for one-clock timed automata, and undecidable in general, even for positive frequency condition.

The corner-point abstraction to abstract timing aspects with rewards The region abstraction is not suitable to study the notion of frequencies in a timed automaton because the timing information is removed by the abstraction. However, a refinement of the region abstraction has been introduced in [BBL08], which allows to preserve some information about time elapsing along runs. This corner-point abstraction is central in our study of the set of frequencies in timed automata.

The article [BBL08] proposes an algorithm to find an optimal scheduling for timed automata with both costs and rewards. More precisely, the model is timed automata equipped with cost and reward functions associating with each location and each edge a cost and a reward. The semantics is a transition system with costs and rewards. The cost and the reward of a discrete transition are simply the cost and the reward of the corresponding edge. In a location whose cost and reward are respectively c and r, the cost and the reward of a delay transition of d time units are respectively $c.d$ and $r.d$. The ratio of a run is then the limit sup of the ratios of the accumulated costs over the accumulated rewards.

Then, the goal is to find a run with optimal ratio (minimum) or an optimal family (*i.e.* a sequence of runs whose ratios tend to the infimum) if there is no optimal run. To do so, the authors introduce the corner-point abstraction, an extension of the region automaton storing time information. Regions are associated with one of their extremal points, called corners. Roughly, dense-time transitions in the timed automaton are abstracted by discrete-time transitions with integer values corresponding to rewards. For example, assuming a single clock x, the successor of the region $0 < x < 1$ associated with the corner $x = 0$ is the region $0 < x < 1$ with the corner $x = 1$. Such a transition corresponds to an abstract time (or reward) 1. However, the successor of $0 < x < 1$ with the corner $x = 1$ is the region $x = 1$ with its unique corner $x = 1$. This transition has reward 0. Moreover, costs 1 are put on transitions with rewards 1 and for which the sources locations are accepting. Costs and rewards are thus translated in the abstraction in a natural way. The problem then reduces to finding a cycle with minimal ratio in the corner-point abstraction, and try to mimic it in the timed automaton. Imprecisions in the lifting potentially lead to the construction of an optimal family rather that an optimal run.

Contribution: frequencies in one-clock timed automata Adding quantitative aspects to timed automata often comes with a cost in terms of decidability and complexity. All along this part, we are looking for reasonable restrictions on the timing behaviors of the system to get positive results about

the computation of the set of frequencies in a timed automaton, or at least of its bounds.

- **General framework** The general framework consists in adapting the corner-point abstraction of [BBL08] to the study of frequencies. The abstraction of dense-time transitions by discrete-time transitions allows one to derive a notion of abstract frequency (called ratio) for the runs in the corner-point abstraction. We thus study the set of ratios in this simpler model. Then, the goal is to find sufficiently tight relations between runs in the timed automaton and runs in its corner-point abstraction, in order to translate results from the abstraction to the timed automaton. The model of [BBL08] is a generalization of timed automata with frequencies. Our goal is to extend the existing results developing stronger links between timed automata and their corner-point abstractions.

- **Techniques for one-clock timed automata** We first develop techniques over one-clock timed automata, a restrictive model which is already expressive, but have relatively simple timed behaviors. Under this restriction, we are able to deal with Zeno runs, whereas frequencies of Zeno runs have very different properties from the non-Zeno case. Indeed, if the accumulated delay along a run is not bounded, finite prefixes can be neglected in the computation of the proportion of time elapsed in accepting locations, in the same way as the first elements of a sequence can be neglected to compute the limit. In other words, the divergence of time allows to forget finite prefixes whatever their length. Note that in [BBL08], timed automata are assumed to be strongly non-Zeno (*i.e.* there is no Zeno run).

- **Convergence phenomena** Unfortunately, techniques used for one-clock timed automata do not extend to two-clock timed automata. Nevertheless, the examples illustrating these limitations share a common character: automata contain cycles along which some convergence phenomena are forced. More precisely, there are convergences between clocks' values over all runs which iterate endlessly such a cycle. This observation led us to a restriction to timed automata without such convergences, which is realistic from an implementability point of view, because they would need unbounded precision for clocks to be observed. A way to detect these convergences along a cycle has been introduced in [BA11] to characterize timed automata whose entropy is positive.

The notion of forgetfulness [BA11] In [BA11], other quantitative aspects are studied: volume and entropy of timed languages. The volume of a timed language L_n, of timed words of length n, is naturally defined as the sum over the untimed words w, of length n, of the volumes of the polyhedra in $\mathbb{R}^n$ corresponding to the possible sequences of delays associated to w in L_n. Roughly, the volume of languages read in a timed automaton restricted to words of length n grows exponentially with n. The entropy of the language of a timed automaton is thus defined as the exponential rate. Some timed automata have a degenerated entropy $-\infty$ (due to punctual guards for example). The paper presents a characterization of timed automata whose language has a non-degenerated (positive) entropy.

The solution is more important to us than the problem solved, and represents a notable progress in the understanding of the convergence phenomena along infinite runs in timed automata. Indeed, the famous notion of zenoness does not cover all the convergence cases. For example, along a cycle, delays in a fixed location can be forced to be smaller and smaller, as first exhibited in [CHR02]. These convergences produce a degenerated entropy. It is proved in [BA11] that a timed automaton has a positive entropy if and only if it contains a reachable cycle without such convergences. Such a cycle is said to be *forgetful*, because delays at a given moment only limitedly impact over future delays, and will be "forgotten". A way to decide whether a cycle is forgetful, is thus proposed.

In particular, our examples of timed automata with several clocks for which the techniques for one-clock timed automata do not apply, contain non-forgetful cycles. Non-forgetfulness in timed automata thus makes some approaches to compute the set of frequencies unfeasible. This is the motivation to firstly consider only timed automata with a single clock where convergence phenomena are restricted to zenoness. Thanks to forgetfulness, the extension to timed automata with several clocks is then possible. We introduce a weaker notion of forgetfulness in order to obtain a class of timed automata for which we can compute the set of frequencies, and whose membership is decidable.

Note that this assumption has been recently used for the synthesis of robust controllers [SBMR13]. Moreover, convergence phenomena seem also to be a cause for the limitation to one-clock stochastic timed automata [BBB$^+$08].

Contribution: extension to timed automata with several clocks In a second phase of our approach, we then propose to compute the set of the frequencies in timed automata excluding convergences, that is in timed automata which are strongly non-Zeno and whose all cycles are forgetful. The membership problem for this class is still an open problem, but we extend the result to strongly non-Zeno timed automata whose simple cycles and their powers (*i.e.* multiple concatenations with themselves) are forgetful, and prove the decidability of the membership problem for this class using the corner-point abstraction.

As for one-clock timed automata, the approach is based on the corner-point abstraction. We prove that the set of ratios in the corner-point abstraction is equal to the set of frequencies in the timed automaton. To do so, we develop technical lemmas which use forgetfulness to lift runs of the corner-point in the timed automaton improving the precision along the run. Intuitively, valuations along a run in the timed automaton are closer and closer to corners along a run in the abstraction. The precision can increase rapidly enough to obtain a run in the timed automaton whose frequency is equal to the ratio of the run in the corner-point abstraction. This is in particular due to the divergence of the accumulated delays along the runs, by assumption. Indeed, it allows to forget the imprecisions in the finite prefixes in the same way as the first element of a sequence does not impact on the cluster points. Our result improves the results of [BBL08], allowing, for example, to ensure the construction of an optimal run (and not only optimal family) assuming the forgetfulness of simple cycles.

Outline This part is structured as follows. Chapter 5 is devoted to the formal definition of the main notions used in the other sections, such as frequency, corner-point abstraction and forgetfulness. Chapters 6 and 7 deal respectively with one-clock timed automata and strongly non-Zeno timed automata whose cycles are forgetful. Finally, the last chapter starts drawing the consequences for the complexity of the emptiness problem and the universality problem in the case of deterministic timed automata. Then it presents the undecidability of the universality problem in general, and the decidability for the particular case of one-clock timed automata for Zeno runs.

Chapter 5

Preliminaries: Frequencies, Corner-Point Abstraction and Forgetfulness

Introduction

We propose new semantics for timed automata based on the proportion of time spent in critical states (called the *frequency*). Contrary to probabilities or volumes [AD09a, AD09b, AD10] that assign a value to sets of behaviors of a timed automaton, the notion of frequency associates a real value (in $[0, 1]$) with each execution of the system. More precisely, the frequency of a run is the proportion of time which is elapsed in accepting locations. It can thus be used in a language-theoretic approach to define quantitative languages associated with a timed automaton, or boolean languages based on quantitative criteria *e.g.*, one can consider the set of timed words for which there is an execution of frequency greater than a threshold λ.

In order to study the set of possible frequencies in a timed automaton we use a refinement of the region abstraction called the *corner-point abstraction* [BBL08]. The idea is to consider each region together with one of its extremal points to add an abstraction of the time along executions. Extremal points, called *corners*, can be seen as abstract valuations, and rewards can be put over the transitions going from a corner to another one, abstracting the time elapsing. Thanks to this notion of abstract time, we define an abstraction of the frequency for the execution in the corner-point abstraction, called *ratio*, as the proportion of rewards in accepting states.

Adding quantitative aspects to timed automata often comes with a cost (in terms of decidability and complexity), and it is often required to restrict the timing behaviors of the system to get some computability results.The tradeoff between expressivity and tractability leads us, in a first step, to restrict to one-clock timed automata. Unfortunately, techniques developed for one-clock timed automata, do not extend to timed automata with two clocks or more. Indeed, some convergence phenomena can appear and make the study of frequencies more complex. These convergences are not realistic from an implementability point of view, because they would require an infinite precision of the clocks to be observed. Then, in order to deal with timed automata with several clocks, we consider timed automata without such phenomena. To do so, we define in this chapter the notions of forgetfulness and aperiodicity to characterize timed automata without convergences. The first notion of forgetfulness had been introduced in [BA11] to characterize cycles without such phenomena, thanks to the orbit graph abstraction. We propose a finer definition, based on the corner-point abstraction itself, which is more adapted to our study, and we compare both definitions.

This chapter is devoted to the presentation of four central notions used in the sequel of the part:

the frequency, a frequency-based semantics, the corner-point abstraction and forgetfulness. More precisely, we first introduce the notion of frequency of a run in a timed automaton, and illustrate it by examples. Section 5.2 is devoted to the definition of a frequency-based semantics, and its comparison with the usual Büchi semantics. Section 5.3 is structured as follows. We present the corner-point abstraction, together with the notion of ratio which abstracts frequencies. Then, we give some preliminary results about the computation of the infimum and the supremum values of ratios in the corner-point abstraction. Finally, in Section 5.4, we use the corner-point abstraction to define subclasses of forgetful timed automata.

5.1 Frequencies in timed automata

We consider timed automata in which runs are infinite and, for simplicity, delays are necessarily positive. Results do extend to timed automata allowing zero delays, but case inspections would be even more tedious in this broader framework.

Let us define the central notion of frequency as the proportion of time elapsed in accepting location along a run.

Definition 5.1 (Frequency)**.** *Given $\mathcal{A} = (L, L_0, F, \Sigma, X, E)$ a timed automaton and $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \xrightarrow{\tau_1, a_1} (\ell_2, v_2) \cdots$ an infinite run of $\mathcal{A}$, the* frequency *of F along ϱ, denoted $\mathsf{freq}_{\mathcal{A}}(\varrho)$, is defined as:*

$$\limsup_{n \to \infty} \frac{\sum_{\{i \leq n \mid \ell_i \in F\}} \tau_i}{\sum_{i \leq n} \tau_i}.$$

Note that the limit may not exist. We choose limit sup to have existence of value, but limit inf would be as relevant.

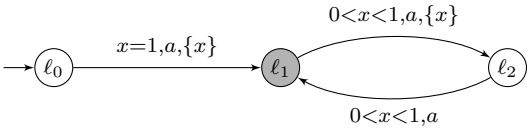


Figure 5.1: A timed automaton to illustrate the notion of frequency.

Illustration of the notion of frequency. Let us consider the timed automaton of Figure 5.1, whose only accepting location is ℓ_1 (accepting locations are colored in gray), to illustrate the notion of frequency. Let ϱ_1, ϱ_2 and ϱ_3 be three runs defined as follows:

$$\varrho_1 = (\ell_0, 0) \xrightarrow{1,a} (\ell_1, 0)\big(\xrightarrow{\frac{1}{3},a} (\ell_2, 0) \xrightarrow{\frac{1}{3},a} (\ell_1, \tfrac{1}{3})\big)^{\omega}$$

$$\varrho_2 = (\ell_0, 0) \xrightarrow{1,a} (\ell_1, 0)\Big(\xrightarrow{\frac{1}{2^k},a} (\ell_2, 0) \xrightarrow{\frac{1}{2^k},a} (\ell_1, \tfrac{1}{2^k})\Big)_{k \geq 1}$$

$$\varrho_3 = (\ell_0, 0) \xrightarrow{1,a} (\ell_1, 0)\Big(\big(\xrightarrow{\frac{1}{3},a} (\ell_2, 0) \xrightarrow{\frac{1}{6},a} (\ell_1, \tfrac{1}{6})\big)^{2^{2^{2k}}} \big(\xrightarrow{\frac{1}{6},a} (\ell_2, 0) \xrightarrow{\frac{1}{3},a} (\ell_1, \tfrac{1}{3})\big)^{2^{2^{2k+1}}}\Big)_{k \geq 1}.$$

- Run ϱ_1 ends by alternating delays one third in both locations of the cycle. The accumulated time along this run diverges, hence whatever the length of the prefix $(\ell_0, 0) \xrightarrow{1,a} (\ell_1, 0)$, it is

neglected in the computation of the frequency. Indeed, it can be expressed under the following form: $\limsup_{k\to\infty} \frac{C+k*\frac{1}{3}}{D+k*\frac{2}{3}}$. The frequency is thus $\frac{1}{2}$.

- Run ϱ_2 ends by alternating the same delay in both locations of the cycle, nevertheless, its frequency is not $\frac{1}{2}$. Indeed, the convergence of the time along the run forces to take into account the prefix. The run starts with a delay 1 in a non-accepting location before the infinite suffix where delays are the same in accepting and non-accepting locations and whose accumulated delay is 2. The frequency is thus $\frac{0+1}{1+2} = \frac{1}{3}$.

- Run ϱ_3 ends by alternating phases where the proportion of time elapsed in accepting locations are different. The length of the phases grows doubly exponentially which implies the non-convergence of the proportion of time elapsed in the accepting location. Indeed, in the first phase, the proportion of time elapsed in the accepting location is $\frac{2}{3}$ and in the second phase, it is $\frac{1}{3}$. Let us prove that $\frac{2}{3}$ is a cluster point of the sequence of the proportions of time elapsed in the accepting location along ϱ_3. To do so, consider the sequence $(h_k)_{k\geq 1}$ of the proportions at the moment where one goes from the first phase to the second one.

$$\begin{aligned} h_k &= \frac{\left(\sum_{i=1}^{k-1} \frac{2^{2^{2i}}}{3} + \frac{2^{2^{2i+1}}}{6}\right) + \frac{2^{2^{2k}}}{3}}{1+\left(\sum_{i=1}^{k-1} \frac{2^{2^{2i}}}{2} + \frac{2^{2^{2i+1}}}{2}\right) + \frac{2^{2^{2k}}}{2}} \geq \frac{\frac{1}{6}\left(\sum_{i=1}^{2k-1} 2^{2^i}\right) + \frac{2^{2^{2k}}}{3}}{1+\frac{1}{2}\left(\sum_{i=1}^{2k-1} 2^{2^i}\right) + \frac{2^{2^{2k}}}{2}} \\ &\geq \frac{\frac{2^{2^{2k-1}}}{6} + \frac{2^{2^{2k}}}{3}}{1+2^{2^{2k-1}} + \frac{2^{2^{2k}}}{2}} \geq \frac{\frac{1}{2^{2^{2k-1}}} + 2}{\frac{6}{2^{2^{2k}}} + \frac{6}{2^{2^{2k-1}}} + 3} \end{aligned}$$

  Moreover, $h_k \leq \frac{2}{3}$ for all k, then $\frac{2}{3}$ is a cluster point of the sequence of the proportions of time elapsed in the accepting location along ϱ_3. In the same way with the other switch of phase, one can prove that $\frac{1}{3}$ is also a cluster point. The sequence is clearly upper-bounded by $\frac{2}{3}$, hence it is the largest cluster point. As a consequence, the frequency of ϱ_3 is $\frac{2}{3}$.

In the sequel, given a timed automaton $\mathcal{A}$, we aim at computing the set of frequencies of the infinite runs of $\mathcal{A}$. To do so, we sometimes distinguish the Zeno and non-Zeno runs of $\mathcal{A}$.

Notation 5.1.

- $\mathsf{Freq}(\mathcal{A})$ *denotes the set of frequencies of infinite runs of* $\mathcal{A}$*;*
- $\mathsf{Freq}_Z(\mathcal{A})$ *denotes the set of frequencies of Zeno runs of* $\mathcal{A}$*;*
- $\mathsf{Freq}_{nZ}(\mathcal{A})$ *denotes the set of frequencies of non-Zeno runs of* $\mathcal{A}$*;*

For example, Figure 5.2 represents a timed automaton $\mathcal{A}$ with $F = \{\ell_1\}$, such that $\mathsf{Freq}(\mathcal{A}) = \mathsf{Freq}_{nZ}(\mathcal{A}) = [0, 1[$. Indeed, there is no Zeno run in $\mathcal{A}$ and there is an underlying constraint along the cycle which ensures that delays elapsed in the accepting location are decreasing. This implies that frequencies of an infinite run in $\mathcal{A}$ is of the form $1 - \varepsilon$ with $\varepsilon \in]0, 1]$.

Given a timed automaton, one can build a timed automaton having only region guards while preserving the set of frequencies. In fact, we need to extend the guards with diagonal constraints of the form $x - y \sim c$ where $x, y \in X$, $c \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$, but both models are known to be equivalent [Bou09]. In the sequel, timed automata are thus assumed to be "split in regions", that is all guards are regions, and all the transitions can be fired. Moreover, in order to take into account that zero delays are not allowed in the semantics, transitions with a constraint of the form $x = 0$ are removed and transitions with a punctual constraint (of the form $x = c$) are changed to one targeted at the time-successor (with constraint $x > c$) if x is not reset.

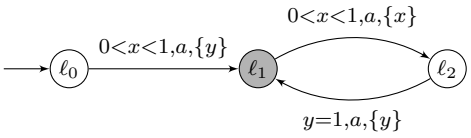


Figure 5.2: A timed automaton $\mathcal{A}$ to illustrate the different sets of frequencies.



(a) $\mathcal{A}_1$. (b) $\mathcal{A}_2$.

Figure 5.3: Examples for the comparison between universality problems.

5.2 Frequency-based semantics

In this section, we define frequency-based acceptances for timed automata, the emptiness and universality problems and we compare the positive-frequency acceptance to the Büchi acceptance.

5.2.1 Frequencies and timed automata

Definition 5.2 (Acceptances and languages)**.**

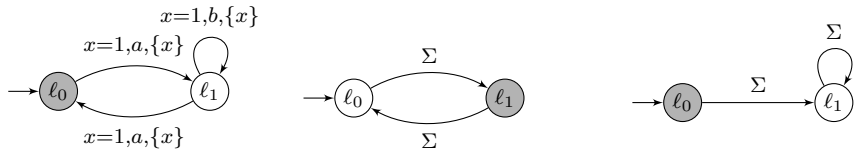
- *A timed word w is said* accepted by $\mathcal{A}$ with positive frequency *if there exists a run ϱ which reads w and such that* $\mathsf{freq}_{\mathcal{A}}(\varrho)$ *is positive.*
- *The* positive-frequency language *of $\mathcal{A}$, noted $\mathcal{L}_{>0}(\mathcal{A})$, is the set of timed words that are accepted with positive frequency by $\mathcal{A}$.*
- *A timed word w is said* accepted by $\mathcal{A}$ under $\sharp\lambda$*, a frequency-based constraint with $\sharp \in \{<, >, \leq, \geq\}$ and $\lambda \in [0,1]$, if there exists a run ϱ which reads w and such that* $\mathsf{freq}_{\mathcal{A}}(\varrho)\sharp\lambda$.
- *The* language under $\sharp\lambda$ *of $\mathcal{A}$, noted $\mathcal{L}\sharp\lambda$, is the set of timed words that are accepted with this constraint by $\mathcal{A}$.*

In this part, we focus on the study of the set of frequencies of runs in a timed automaton, with the motivation to decide classical language problems.

Definition 5.3 (Emptiness problem)**.** *The* emptiness problem *asks, given a timed automaton $\mathcal{A}$ and a frequency-based constraint $\sharp\lambda$, whether $\mathcal{L}\sharp\lambda$ is empty.*

Definition 5.4 (Universality problem and variants)**.** *The* universality problem for infinite (resp. non-Zeno, Zeno) timed words *asks, given a timed automaton $\mathcal{A}$ over the alphabet Σ and a frequency-based constraint $\sharp\lambda$, whether all (resp. all non-Zeno, all Zeno) timed word belong to $\mathcal{L}\sharp\lambda$.*

Let us explain why the universality problems with positive-frequency acceptance are not comparable when considering Zeno timed words or non-Zeno timed words. Both timed automata of Figure 5.3 illustrate this. The timed automaton $\mathcal{A}_1$ is universal for Zeno timed words but is not non-Zeno timed words. Indeed, $\mathcal{A}_1$ does not accept $(a, 2n)_{n\geq 1}$. In the same way, $\mathcal{A}_2$ is not universal for Zeno timed words, whereas it is for non-Zeno timed words Indeed, $\mathcal{A}_2$ does not accept $(a, \sum_{i=2}^{n} \frac{1}{2^i})_{n\geq 1}$.



(a) Expressiveness. (b) Universality (non-Zeno). (c) Universality (Zeno).

Figure 5.4: Automata for the comparison with the usual semantics.

5.2.2 A brief comparison with the usual semantics

The usual semantics for timed automata considers a Büchi acceptance condition. We naturally explore differences between this usual semantics, and the closest one we introduced, that is the positive-frequency acceptance. The expressiveness of timed automata under those acceptance conditions is not comparable, as witnessed by the automaton represented in Figure 5.4(a): on the one hand, its positive-frequency language is not timed-regular (*i.e.* accepted by a timed automaton with a standard Büchi acceptance condition), and on the other hand, its Büchi language cannot be recognized by a timed automaton with a positive-frequency acceptance condition.

Indeed, the language $\mathcal{L}_B$ accepted with the Büchi semantics contains, for all $N \in \mathbb{N}$, the word w_N whose untimed word is $a.b^{N+1}.a.a.b^{2(N+1)}.a.a.....b^{k(N+1)}.a.a....$ and delays are exactly one time unit between each action. Let us assume that there exists a timed automaton accepting $\mathcal{L}_B$ with the positive-frequency acceptance, and let N be the number of its locations. Then, there necessarily is a reachable cycle in this automaton with an accepted location and only b as action for the edges, otherwise, the frequency of w_N would be 0. As a consequence, the timed automaton can accept a timed word with a finite number of a's, simply iterating infinitely this cycle.

On the other side, the language $\mathcal{L}_{>0}$ accepted with positive frequency contains, for all $N \in \mathbb{N}$, the word w'_N whose untimed word is $a.b^{N+1}.a.a.b^{N+1}.a.a.....b^{N+1}.a.a....$ and delays are exactly one time unit between each action. Let us assume that there exists a timed automaton accepting $\mathcal{L}_{>0}$ with the Büchi acceptance, and let N be the number of its locations. Then, each factor b^{N+1}, is read through a cycle in this automaton with only b as action for the edges. Then, iterating these cycles, one obtain that a word $w"_N$ whose untimed word is $a.b^{n_0}.a.a.b^{n_1}.a.a.....b^{n_k}.a.a....$, delays are exactly one time unit between each action, and such that for all $k \in \mathbb{N}$, $n_k > k(N+1)$, is also accepted. But this timed word does not belong to $\mathcal{L}_{>0}$.

Beyond their relative expressivity, one can wonder how the notions of universality under Büchi semantics and positive frequency semantics compare. On the one hand, clearly enough, a (non-Zeno-)universal timed automaton with a positive-frequency acceptance condition is (non-Zeno-)universal for the classical Büchi-acceptance. The timed automaton of Figure 5.4(b) is a counterexample to the converse. For instance, let us consider the word alternating delays 1 and delays $\frac{1}{2^n}$ with n the number of transitions already fired. The accumulated delay in the accepting location is bounded when the accumulated delay in non-accepting locations is not. The frequency is thus equal to 0. Then this word is not accepted for the positive frequency semantics, whereas this timed automaton is clearly universal for the Büchi semantics. On the other hand, a Zeno-universal timed automaton under the classical semantics is necessarily Zeno-universal under the positive-frequency acceptance condition, but the automaton depicted in Figure 5.4(c) shows that the converse does not hold. Indeed, the accumulated delay along any Zeno run is finite, then it is sufficient to visit only one accepting location to have a positive frequency, whereas this does not suffice to satisfy the Büchi condition.

5.2.3 A particular case of double-priced timed automata

Timed automata with frequencies can be seen as a particular case of double-priced timed automata of [BBL08]. Let us recall the definition of double-priced timed automata in order to formalize this idea. Note that we added action labels over edges. It does not impact on the results of [BBL08] which only consider costs and ratios of runs.

Definition 5.5 (Double-priced timed automata)**.** *A* double-priced timed automaton *over a set of clocks X is a tuple $(L, L_0, \Sigma, E, \mathsf{c}, \mathsf{r})$, where L is a finite set of locations, ℓ_0 is the initial location, $E \subseteq L \times G_M(X) \times \Sigma \times 2^X \times L$ is the set of edges, and $c, r : (L \cup E) \to \mathbb{Z}$ assign price-rates to locations and prices to edges.*

The semantics of a double-priced timed automaton is a double-priced transition system $(S, s_0, \to, \mathsf{cost}, \mathsf{reward})$ over X, where $S = L \times \mathbb{R}_+^X$, $s_0 = (\ell_0, 0^X)$, and $\to$ is defined as follows: for every state $(\ell, v) \in S$,

- for every $\tau \in \mathbb{R}_+$, $((\ell, v), \tau, (\ell, v + \tau)) \in \to$ (written $(\ell, v) \xrightarrow{\tau} (\ell, v + \tau)$ for short);
- for every edge (ℓ, g, X', ℓ'), $((\ell, v), (\ell', v')) \in \to$ if $v \models g$ and $v' = v_{[X' \leftarrow 0]}$ (written $(\ell, v) \to (\ell, v)$ for short);.

Moreover, cost and reward are respectively defined for these transitions as follows:

- $\mathsf{cost}((\ell, v), \tau, (\ell, v + \tau)) = \mathsf{c}(\ell) * \tau$ and $\mathsf{reward}((\ell, v), \tau, (\ell, v + \tau)) = \mathsf{r}(\ell) * \tau$;
- $\mathsf{cost}((\ell, v), g, X', (\ell', v')) = \mathsf{c}(\ell, g, X', \ell')$ and $\mathsf{reward}((\ell, v), (\ell', v')) = \mathsf{r}(\ell, g, X', \ell')$.

A run is an infinite sequence of consecutive moves. Without loss of generality, one can assume that a run alternates delays and discrete transitions and for readability, we write $(\ell, v) \xrightarrow{\tau, a} (\ell, v'')$ for $(\ell, v) \xrightarrow{\tau} (\ell, v') \xrightarrow{a} (\ell, v'')$ and the cost (resp. reward) of the double transition is simple the sum of the costs (resp. rewards) of both transitions. The ratio of a run is the ratio of the accumulated costs over the accumulated rewards along the run.

Definition 5.6 (Ratio)**.** *Given $\mathcal{A} = (L, L_0, E, \mathsf{c}, \mathsf{r})$ a timed automaton and $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \xrightarrow{\tau_1, a_1} (\ell_2, v_2) \cdots$ an infinite run of $\mathcal{A}$, the* ratio *of ϱ, denoted $\mathsf{Rat}(\varrho)$, is defined as:*

$$\limsup_{n \to \infty} \frac{\sum_{i \leq n} \mathsf{cost}((\ell_i, v_i) \xrightarrow{\tau_i, a_i} (\ell_{i+1}, v_{i+1}))}{\sum_{i \leq n} \mathsf{reward}((\ell_i, v_i) \xrightarrow{\tau_i, a_i} (\ell_{i+1}, v_{i+1}))}.$$

This notion is similar to the frequency of a run in timed automata. More precisely, timed automata with frequencies are a particular case of double-priced timed automata. Indeed, given a timed automaton with frequencies $\mathcal{A} = (L, L_0, F, \Sigma, X, M, E)$, one can build a double-priced automaton $\mathcal{A}' = (L, L_0, \Sigma, E, \mathsf{c}, \mathsf{r})$ with the same set of runs and such that the frequency of any run in $\mathcal{A}$ is equal to its ratio in $\mathcal{A}'$. To do so, we just have to define c and r taking into account F. Costs and rewards over edges are useless to compute frequencies, we then set them to 0. For locations, we simply put rewards 1 everywhere and costs 1 over the locations of F and 0 otherwise. As a consequence, in the definition of the ratio, $\mathsf{cost}((\ell_i, v_i) \xrightarrow{\tau_i, a_i} (\ell_{i+1}, v_{i+1})) = \tau_i$ if ℓ_i belongs to F and $\mathsf{cost}((\ell_i, v_i) \xrightarrow{\tau_i, a_i} (\ell_{i+1}, v_{i+1})) = 0$ otherwise; and $\mathsf{reward}((\ell_i, v_i) \xrightarrow{\tau_i, a_i} (\ell_{i+1}, v_{i+1})) = \tau_i$ for every location. We thus obtain $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\varrho)$ for every ϱ.

In the sequel, we thus use some results of [BBL08] for timed automata with frequencies. As in [BBL08], we use the corner-point abstraction to study ratios of runs in a simpler model.

5.3 The corner-point abstraction

In this section, we present the corner-point abstraction [BBL08] which allows to abstract time in a finer way than in the region abstraction and thus to define the notion of ratio of a run in the corner-point abstraction similarly to frequency in the timed automaton. This yields a simpler model to study in order to obtain information about the frequencies in the timed automaton itself. Finally, we give an expression of the set of ratios of runs in the corner-point abstraction whose accumulated abstract delays diverge.

5.3.1 Definition and examples

Let $\mathcal{A} = (L, L_0, F, \Sigma, X, M, E)$ be a timed automaton. The corner-point abstraction is a refinement of the region abstraction, where each state is composed of a region with one of its extremal points. Recall that a region R' is a *time-successor* of a region R if there exists $v \in R$ and $t \in \mathbb{R}_+$ such that $v + t \in R'$ and $R' \neq R$. The set of the time-successors of a region is naturally ordered, let us define the mapping $timeSucc : Reg_{\mathcal{A}} \to Reg_{\mathcal{A}}$ which associates with any region, its first time-successor. The particular case of the region $\{\perp^X\}$ where all the clocks are larger than M is fixed as follows : $timeSucc(\{\perp^X\}) = \{\perp^X\}$.

Moreover, given a region R, $\alpha \in (\mathbb{N}_{\leq M} \cup \perp)^X$ is a *corner* of X if for all clock x such that $R_{|\{x\}} \subseteq [0, M]$, $\alpha(x)$ is in the closure of $R_{|\{x\}}$ for the usual topology over $\mathbb{R}$, and for all clock x such that $R_{|\{x\}} =]M, +\infty[$, $\alpha(x) = \perp$. Thus, an *$\mathcal{A}$-pointed region* (pointed region for short) is a pair (R, α) where R is a region and α a corner of R. The set of $\mathcal{A}$-pointed regions is written $Reg\bullet_{\mathcal{A}}$. The operations defined on the valuations of a set of clocks are extended in a natural way to the corners, with the convention that $M + 1 = \perp$ and $\perp + 1 = \perp$. Then $timeSucc$, the function giving the immediate time successor can be extended to pointed regions:

$$timeSucc(R, \alpha) = \begin{cases} (R, \alpha + 1) & \text{if } \alpha + 1 \text{ is a corner of } R \\ (timeSucc(R), \alpha') & \text{otherwise} \end{cases}$$

where $\forall x,\ \alpha'(x) = \alpha(x)$ if x is bounded in $timeSucc(R)$ and otherwise $\alpha'(x) = \perp$.

Using this mapping, the construction of the corner-point abstraction is very similar to the usual region automaton.

Definition 5.7 (Corner-point abstraction)**.** *The* corner-point abstraction *of $\mathcal{A}$ (corner-point of $\mathcal{A}$ for short) is the finite automaton $\mathcal{A}_{cp} = (L_{cp}, L_{0,cp}, F_{cp}, \Sigma_{cp}, E_{cp})$ where $L_{cp} = L \times Reg\bullet_{\mathcal{A}}$ is the set of states, $L_{0,cp} = L_0 \times \{(\{\overline{0}\}, \overline{0})\}$ is the set of initial states, $F_{cp} = F \times Reg\bullet_{\mathcal{A}}$ is the set of accepting states, $\Sigma_{cp} = \Sigma \cup \{\varepsilon\}$, and $E_{cp} \subseteq L_{cp} \times \Sigma_{cp} \times L_{cp}$ is the finite set of edges defined as the union of discrete transitions and idling transitions:*

- discrete transitions: $(\ell, (R, \alpha)) \xrightarrow{a} (\ell', (R', \alpha'))$ *if there exists a transition* $\ell \xrightarrow{g,a,X'} \ell'$ *in $\mathcal{A}$, such that* $R = g$[1] *and* $(R', \alpha') = (R_{[X' \leftarrow 0]}, \alpha_{[X' \leftarrow 0]})$,
- idling transitions: $(\ell, (R, \alpha)) \xrightarrow{\varepsilon} (\ell, (R', \alpha'))$ *if* $(R', \alpha') = timeSucc(R, \alpha)$.

In particular, as a consequence of $\perp + 1 = \perp$, there is an idling loop on each state whose pointed region is $((M, +\infty)^X, \perp^X)$. For example, Figure 5.5 represents the corner-point abstraction of the timed automaton in Figure 5.1 where we use the following convention to represent pointed regions:

[1] Recall that guard of timed automata are assumed to be regions.

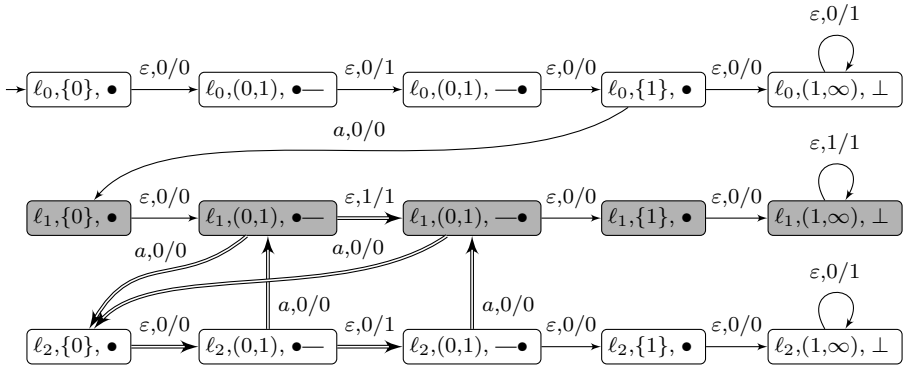


Figure 5.5: The corner-point abstraction $\mathcal{A}_{cp}$ of $\mathcal{A}$ represented Figure 5.1.

- $(k, k+1)$, •— represents $((k, k+1), k)$,
- $(k, k+1)$, —• represents $((k, k+1), k+1)$,
- $\{k\}$, • represents $(\{k\}, k)$.

In the sequel, we use the natural extension of this convention with two clocks.

We are now able to link runs of a timed automaton and runs in its corner-point abstraction by the projection. A run in a timed automaton admits a natural projection in the region automaton. In the corner-point abstraction, several runs may correspond to a run in the timed automaton. This is due to the possibility, in the abstraction to choose to stay in the first corner of the region or to go to the second one before firing a discrete transition. For instance in the corner-point abstraction of Figure 5.5, to read a from $(\ell_1, (0,1))$, one can choose between corners 0 and 1, whatever the run which is abstracted. As a consequence, we define the projection of a run as the set of all its possible abstraction in the corner-point abstraction.

One can distinguish two types of idling transitions, the transitions which change the current region, but not the current corner and the transitions which do not change the current region, but change the corner. Informally, they respectively correspond to abstract delays 0 and 1 considering corners as kinds of abstract valuations. Let us define formally the notion of abstract valuations along runs in the corner-point abstraction. The idea is to count the number of idling transitions which do not change the current region since the last resets of the clocks.

Definition 5.8 (Abstract valuation)**.** *Let $\pi = (\ell_0, \{0^X\}, \bullet) \xrightarrow{b_0} (\ell_1, R_1, \alpha_1) \xrightarrow{b_1} \cdots$ a run in a corner-point abstraction of a timed automaton. The i-th* abstract valuation $\pi[i] : X \to \mathbb{N}^X$ *along π is defined as follows: $\pi[i](x)$ is the number of idling transitions of the form $(\ell, R, \alpha) \xrightarrow{\varepsilon} (\ell, R, \alpha + 1)$ between the $i+1$-th discrete transition and the previous reset of x.*

Note that if the clock x in the pointed region (R_i, α_i) before the $i+1$-th discrete transition is bounded, then $\pi[i](x) = \alpha_i(x)$. This notion simply allows to obtain a finer notion of projection. Indeed, we thus avoid to project runs with a delay of 3000 time units in an unbounded regions over the same runs than the same run with a delay 3 instead of 3000. We are going to link the abstract valuations along runs in the corner-point abstraction with valuations along runs in the timed automaton, noted $\varrho[i]$ and defined as the sum of the valuation of the $i+1$-th state of ϱ and the $i+1$-th delay. In other

words, we consider the valuations just before reading actions. Formally, given $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0,a_0} (\ell_1, v_1) \xrightarrow{\tau_1,a_1} \cdots$, $\varrho[n] = v_n + \tau_n$. Let us now define the projection of a run.

Definition 5.9 (Projection). *The* projection *of a (finite or infinite) run $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0,a_0} (\ell_1, v_1) \xrightarrow{\tau_1,a_1} \cdots$ of $\mathcal{A}$, denoted by $\mathsf{Proj}(\varrho)$, is the set of runs π of $\mathcal{A}_{cp}$ such that for all $i \in \mathbb{N}$, the $i+1$-th discrete transition goes from a state $(\ell_i, R(\varrho[i]), \alpha)$ to a state $(\ell_{i+1}, R(v_{i+1}), \alpha')$ and for all clocks $x \in X$, the number $\pi[i](x)$ has to be equal to $\lfloor \varrho[i](x) \rfloor$ or $\lceil \varrho[i](x) \rceil$.*

Let us illustrate these two notions over the beginning of run ϱ_1 of the timed automaton of Figure 5.1. Recall that $\varrho_1 = (\ell_0, 0) \xrightarrow{1,a} (\ell_1, 0) \xrightarrow{\frac{1}{3},a} (\ell_2, 0) \xrightarrow{\frac{1}{3},a} (\ell_1, \frac{1}{3}) \xrightarrow{\frac{1}{3},a} \cdots$. Then $\pi = (\ell_0, \{0\},\ \bullet\ ) \xrightarrow{\varepsilon} (\ell_0, (0,1),\ \bullet\!\!-\ ) \xrightarrow{\varepsilon} (\ell_0, (0,1),\ -\!\!\bullet\ ) \xrightarrow{\varepsilon} (\ell_0, \{1\},\ \bullet\ ) \xrightarrow{a} (\ell_1, \{0\},\ \bullet\ ) \xrightarrow{\varepsilon} (\ell_1, (0,1),\ \bullet\!\!-\ ) \xrightarrow{a} (\ell_2, \{0\},\ \bullet\ ) \xrightarrow{\varepsilon} (\ell_2, (0,1),\ \bullet\!\!-\ ) \xrightarrow{a} (\ell_1, (0,1),\ \bullet\!\!-\ ) \xrightarrow{\varepsilon} (\ell_1, (0,1),\ -\!\!\bullet\ )$ is a prefix of some runs belonging to $\mathsf{Proj}(\varrho_1)$. Indeed, the first discrete transition along π is $(\ell_0, \{1\},\ \bullet\ ) \xrightarrow{a} (\ell_1, \{0\},\ \bullet\ )$. This transition is fired from a state with the right location ℓ_0 with the region $x = 1$ which clearly contains the valuation $\varrho[0]$ defined by $\varrho[0](x) = 1$, and the target region $\{0\}$ also contains the corresponding valuation v_1 defined by $v_1(x) = 0$ in $\mathcal{A}$. The abstract valuation $\pi[0]$ is defined by $\pi[0](x) = 1$ because there is a single idling transition before the discrete transition, which does not change the current region. Hence, it satisfies the constraint ($\pi[0](x) = \lfloor 1 \rfloor = 1$ or $\pi[0](x) = \lceil 1 \rceil = 1$). Let us now consider the second discrete transition $(\ell_1, (0,1),\ \bullet\!\!-\ ) \xrightarrow{a} (\ell_2, \{0\},\ \bullet\ )$. Region $(0,1)$ contains the valuation $\varrho[1]$ defined by $\varrho[1](x) = \frac{1}{3}$. Moreover, clock x has been reset at the last discrete transition and there is no idling transition not changing the current region between both discrete transitions, then $\pi[1](x) = 0$. In particular, the constraint ($\pi[1](x) = \lfloor \frac{1}{3} \rfloor = 0$ or $\pi[1](x) = \lceil \frac{1}{3} \rceil = 1$) is satisfied.

Remark that in $\mathcal{A}$ all infinite runs admit the same projection: the set of all the runs which do not reach a region $\perp$ (otherwise discrete transitions are no more enabled).

We saw how to project runs of timed automata in corner-point abstraction, now we introduce a notion which can be seen as a way to quantify, in a sense, the distance between a run in a timed automaton and a run of its projection. Roughly, we say that a run in the timed automaton $\mathcal{A}$ mimics a run of its projection up to ε if the valuations along the run in $\mathcal{A}$ are ε-close to the abstract valuations along the run in the corner-point abstraction.

Definition 5.10 (Mimicking). *Given $\varepsilon > 0$, we say that a (finite or infinite) run ϱ of $\mathcal{A}$* mimics up to *$\varepsilon > 0$ a (finite or infinite) run π in $\mathsf{Proj}(\varrho)$ if, for all indices i, the i-th discrete transition of π goes from a state $(\ell_i, R(\varrho[i]), \alpha)$ such that, for all clock $x \in X$, $|\varrho[i](x) - \pi[i](x)| < \varepsilon$.*

This notion is very important in the rest of the part. It is the key to lift results on the corner-point abstraction to the original timed automaton by building precise mimicking runs of the runs of the corner-point abstraction.

In the sequel we often consider cycles of the graph of $\mathcal{A}$ (cycles of $\mathcal{A}$ for short), that is some sequences $\ell_0 e_0 \ell_1 e_1 \cdots e_{n-1} \ell_n = \ell_0$ such that for all $0 \leq i \leq n-1$ the edge e_i goes from ℓ_i to ℓ_{i+1} in $\mathcal{A}$. Similarly to runs, we define the *projection of a cycle* C of $\mathcal{A}$, denoted by $\mathsf{Proj}(C)$. If C is a simple cycle with no region $\perp^X$, $\mathsf{Proj}(C)$ is the subgraph of $\mathcal{A}_{cp}$ covered by the projection of any finite run of $\mathcal{A}$ along C. For example, the projection of the cycle of the timed automaton Figure 5.1 is the subgraph of its corner-point abstraction whose edges are drawn with double arrows on Figure 5.5. If C is a simple cycle with some regions $\perp^X$, we simply add the idling loops associated with each state of the form $(\ell, \{\perp^X\}, \perp^X)$. To define the projection of a cycle C which is not simple, we first unfold the timed automaton $\mathcal{A}$ to obtain an equivalent simple cycle. For example, the projection of

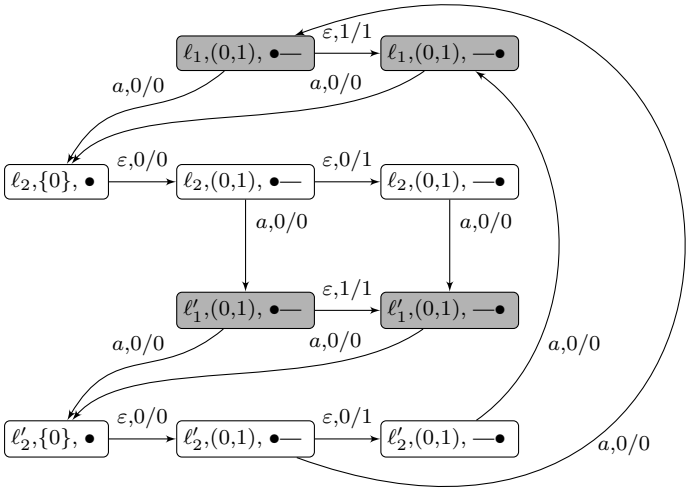


Figure 5.6: Illustration of the projection of a cycle.

the cycle constituted of two iterations of the cycle of the timed automaton Figure 5.1 is represented in Figure 5.6.

5.3.2 Ratios in the corner-point abstraction

In the corner-point abstraction, the idling transitions which do not change the current region correspond to the elapsing of one time unit. These abstract delays are used to abstract the frequencies in a timed automaton by ratios in its corner-point abstraction. To do so, the corner-point is equipped with costs and rewards as follows:

- the *reward* of a transition is 1 if it is of the form $(\ell, R, \alpha) \xrightarrow{\varepsilon} (\ell, R, \alpha')$, and 0 otherwise;
- the *cost* of a transition is 1 if its reward is 1 and the location ℓ is accepting, and 0 otherwise.

Note that in particular, the loops on the states whose region is $\{\bot^X\}$ have reward 1. The accumulated rewards correspond to the abstract time elapses and accumulated costs corresponds to abstract time elapses in accepting locations. In Figure 5.5, costs and rewards are written over the edges as follows: cost / reward. We use the same convention along runs in the sequel. Thanks to these costs and rewards, the *ratio* of an infinite run of the corner-point can be defined, similarly to the frequency in the timed automaton.

Definition 5.11 (Ratio)**.** *Given $\mathcal{A}_{cp} = (L_{cp}, L_{0,cp}, F_{cp}, \Sigma_{cp}, E_{cp})$ a corner-point abstraction of a timed automaton and $\pi = (\ell_0, \{0^X\}, \bullet) \xrightarrow{b_0,c_0/d_0} (\ell_1, R_1, \alpha_1) \xrightarrow{b_1,c_1/d_1} \cdots$ an infinite run of $\mathcal{A}_{cp}$, the* ratio *of F along ϱ, denoted $\mathsf{Rat}(\pi)$, is defined as:*

$$\limsup_{n\to\infty} \frac{\sum_{\{i\leq n\}} c_i}{\sum_{i\leq n} d_i}.$$

An infinite run in the corner-point is said *reward-converging* (resp. *reward-diverging*) if the accumulated reward is finite (resp. infinite). This notion is close to zenoness of runs in timed automata. Yet some Zeno runs may be projected to reward-diverging runs in the corner-point abstraction and, the other way around, non-Zeno runs can be projected to reward-converging runs. For example, the run whose delays along the cycle of Figure 5.1 are all 0.2 can be projected to the run iterating the cycle of the corner-point in Figure 5.5 with only zero rewards.

Notation 5.2.

- $\mathsf{Rat}(\mathcal{A}_{cp})$ *denotes the set of ratios of infinite runs of $\mathcal{A}_{cp}$;*
- $\mathsf{Rat}_{r-c}(\mathcal{A}_{cp})$ *denotes the set of ratios of reward-converging runs of $\mathcal{A}_{cp}$;*
- $\mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$ *denotes the set of ratios of reward-diverging runs of $\mathcal{A}_{cp}$;*

We also say *reward-diverging* for a cycle of $\mathcal{A}_{cp}$ whose accumulated reward is positive and *reward-converging* otherwise.

5.3.3 Set of ratios in the corner-point abstraction

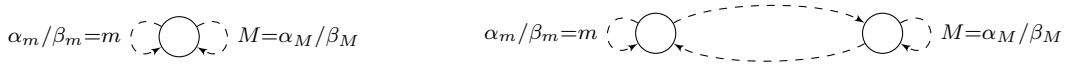
We defined ratios in the corner-point abstraction. In this section, we make the connection with frequencies in the original timed automaton, thus seeing ratios as abstractions of frequencies. The goal is to use this simpler model to obtain results about the frequencies in the timed automaton. In this section, we precisely characterize $\mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$, the set of ratios of reward-diverging runs of the corner-point.

Theorem 5.1. *Let $\{C_1, \cdots, C_k\}$ be the set of reachable strongly connected components (SCC for short) of $\mathcal{A}_{cp}$. Then, $\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}) = \cup_{1\leq i\leq k}[m_i, M_i]$ where m_i (resp. M_i) is the minimal (resp. maximal) ratio for a reward-diverging cycle in C_i.*

As an infinite run necessarily ends in a single SCC, Theorem 5.1 is a straightforward consequence of the following lemma.

Lemma B. *Let C_i be an SCC of $\mathcal{A}_{cp}$. If $\mathcal{R}_i$ denotes the set of ratios of reward-diverging simple cycles in C_i, then the set of ratios of reward-diverging runs of $\mathcal{A}_{cp}$ ending in C_i is the interval $[m_i, M_i]$, where $m_i = \min(\mathcal{R}_i)$ and $M_i = \max(\mathcal{R}_i)$.*

The idea of the proof is that, in C_i, the i-th SCC of $\mathcal{A}_{cp}$, values m_i and M_i are reached by ratios of lasso runs simply iterating respectively two cycles of ratios m_i and M_i in C_i. Then, the intuition is that one can reach all the other values of the interval $[m_i, M_i]$ by alternating both cycles with suitable proportions.



(a) Case 1 (b) Case 2

Figure 5.7: The two possible cases.

Proof. Let π be a reward-diverging run of $\mathcal{A}_{cp}$. We associate with π the SCC C_π of $\mathcal{A}_{cp}$ where π ends up in. First observe that the influence of the prefix leading to C_π is negligible in the computation of the ratio because π is reward-diverging. Precisely, the ratio of the prefix of length n (for n large enough) is of the form:

$$\mathsf{Rat}(\pi_{|n}) = \frac{p_{pref} + P_n}{q_{pref} + Q_n}$$

where p_{pref}/q_{pref} is the ratio of the shortest prefix of π leading to C_π. The sequence Q_n diverges when n tends to infinity because π is reward-diverging. Hence $\lim_{n\to\infty} \mathsf{Rat}(\pi_{|n}) = \lim_{n\to\infty} \frac{P_n}{Q_n}$. As a consequence, without loss of generality, we assume that $\mathcal{A}_{cp}$ is restricted to C_π and π starts in some state of C_π.

Observe now that reward-converging cycles in $\mathcal{A}_{cp}$ necessarily have rewards (and hence costs) 0, and thus do not contribute to the ratio $\mathsf{Rat}(\pi)$. Hence we can assume without loss of generality that π does not visit reward-converging cycles. In the same way as in the proof of [BBL08, Prop. 4], we can decompose π into (reward-diverging) cycles and derive that $\mathsf{Rat}(\pi)$ lies between $m = \min(\mathcal{R}_{C_\pi})$ and $M = \max(\mathcal{R}_{C_\pi})$. Note that the extremal values (m and M) are obtained by a run reaching a cycle with extremal ratio, and iterating it forever.

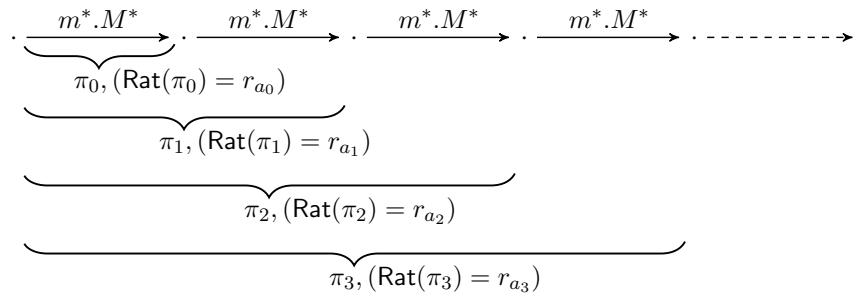
Let us now show that any value in the interval $[m, M]$ is the ratio of some run in $\mathcal{A}_{cp}$ which ends up in the SCC C_π. The arguments are inspired by [CDE$^+$10]. Given $\lambda \in (0, 1)$, we explain how to build a run with ratio $r_\lambda = (1-\lambda)m + \lambda M$. To do so, for $(a_n) \in (\mathbb{Q} \cap (m, M))^{\mathbb{N}}$ a sequence of rational numbers converging to r_λ, we build an run π such that $|\mathsf{Rat}(\pi_{|f(n)}) - a_n| < \frac{1}{n}$ for some increasing function $f \in \mathbb{N}^{\mathbb{N}}$.

Case 1. We first assume for simplicity that in C_π two cycles of respective ratio m and M share a state, as depicted in Figure 5.7(a), and prove a stronger result: we build a run π such that $\mathsf{Rat}(\pi_{|f(n)}) = a_n$. Since two cycles, one of minimal ratio, and the other of maximal ratio share a common state, it suffices to explain how to combine these two cycles to obtain ratio r_λ.

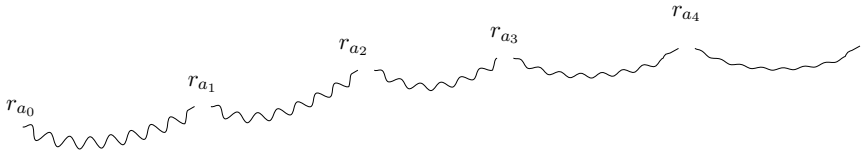
Assume $a_0 = p_0/q_0$ with $(p_0, q_0) \in \mathbb{N}^2$. Let us show how to build a finite run π of ratio $r_{a_0} = (1 - p_0/q_0)m + (p_0/q_0)M$. Assume $m = \alpha_m/\beta_m$ where α_m is the cost of the cycle, and β_m its reward, and similarly $M = \alpha_M/\beta_M$. Taking $(q_0 - p_0)\beta_M$ times the cycle of ratio m and then $p_0\beta_m$ times the cycle of ratio M yields an finite run π_0 with the desired property (this will be $\pi_{|f(0)}$). Indeed:

$$\frac{\big((q_0-p_0)\beta_M\big)\alpha_m + \big(p_0\beta_m\big)\alpha_M}{\big((q_0-p_0)\beta_M\big)\beta_m + \big(p_0\beta_m\big)\beta_M} = \frac{(q_0-p_0)\beta_M\alpha_m + p_0\beta_m\alpha_M}{q_0\beta_M\beta_m} = \frac{q_0-p_0}{q_0}m + \frac{p_0}{q_0}M = r_{a_0}.$$

To build an infinite run with ratio r_λ, we incrementally build prefixes $\pi_{|f(n)}$ (which will be $\pi_{|f(n)}$) of ratio r_{a_n}, starting with π_0, as depicted in the picture below.



Run $\pi_{|f(n+1)}$ has $\pi_{|f(n)}$ as prefix, then iterates the cycle of minimal ratio, and finally iterates the cycle of maximal ratio in order to compensate r_{a_n} and reach ratio $r_{a_{n+1}}$. We assume $a_n = p_n/q_n$ with $(p_n, q_n) \in \mathbb{N}^2$. In $\pi_{|f(n+1)}$ the number of iterations of the cycle of ratio m (resp. the cycle of ratio M) is globally $b_{n+1}(q_{n+1} - p_{n+1})\beta_M$ (resp. $b_{n+1}p_{n+1}\beta_m$) for some $b_{n+1} \in \mathbb{N}_{>0}$. This construction ensures that r_λ is an accumulation point of the set of ratios for the prefixes $\pi_{|f(n)}$. Moreover, since each path fragment starts with iterations of the cycle of minimal ratio first, r_λ is the largest accumulation point of the sequence of the ratios of prefixes after each cycle. The sequence of the ratios of prefixes is sketched below. The oscillations during a cycle become negligible when the length of the run increases. In the picture below, they are represented by shorter and shorter wavelets.



Case 2 In the general case, in the SCC C_π of $\mathcal{A}_{cp}$ the cycles with minimal and maximal ratios do not necessarily share a common state: two finite runs connect the two cycles, as represented on Fig 5.7(b). We fix two cycles of minimal and maximal ratios, and two finite paths π_{mM} and π_{Mm} that connect those cycles in C_π. Similarly to the first case, we show how to build a sequence of finite runs $(\pi_{|f(n)})$ such that each run is the a prefix of the following run and with $|\mathsf{Rat}(\pi_{|f(n)}) - r_{a_n}| < \frac{1}{n}$, and prove that the influence of the finite paths linking the cycles is negligible when n tends to infinity. The run $\pi_{|f(n+1)}$ is defined as the concatenation of $\pi_{|f(n)}$ with π_{Mm} then iterations of the cycle of minimal ratio m then π_{mM} and ending with iterations of the cycle with maximal ratio M. If $\tilde{p}$ and $\tilde{q}$ are respectively the cost and the reward of π_{mM} and π_{Mm} together, then the ratio of $\pi_{|f(n+1)}$ is:

$$\mathsf{Rat}(\pi_{|f(n+1)}) = \frac{b_{n+1}(q_{n+1} - p_{n+1})\beta_M\alpha_m + b_{n+1}p_{a_{n+1}}\beta_m\alpha_M + (n+1)\tilde{p}}{b_{n+1}(q_{n+1} - p_{n+1})\beta_M\beta_m + b_{n+1}p_{a_{n+1}}\beta_m\beta_M + (n+1)\tilde{q}}$$

Since this value tends to $r_{a_{n+1}}$ when b_{n+1} tends to infinity, b_{n+1} can be chosen such that $|\mathsf{Rat}(\pi_{|f(n+1)}) - r_{a_{n+1}}| < 1/(n+1)$. This way, $\lim_{n\to\infty} \mathsf{Rat}(\pi_{|f(n)})$ agrees with $\lim_{n\to\infty} r_{a_n}$, that is $\lim_{n\to\infty} \mathsf{Rat}(\pi_{|f(n)}) = r_\lambda$. The function f is defined by '$f(n)$ is the length of $\pi_{|f(n)}$'. The run π such that, for all n, $\pi_{|f(n)}$ is a prefix of π is unique and have ratio r_λ. □

We thus have a nice expression of the set of ratios of reward-diverging runs in the corner-point abstraction. This set can thus be computed simply inspecting simple cycles of the corner-point abstraction. Unfortunately, we do not have such a general result for the set of ratios of reward-converging runs. As a consequence, we only study bounds of the set of ratios or use restrictions such as forgetfulness.

If we only consider bounds of the set, it is not necessary to consider strongly connected component and there exists a non-deterministic procedure to compute them in logarithmic space in the size of the corner-point abstraction.

Corollary 5.1 (Corollary of Theorem 5.1)**.**

- *If $\mathcal{A}$ has a single clock, there exists a non-deterministic procedure computing* $\inf(\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}))$ *and* $\sup(\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}))$ *in logarithmic space in the size of $\mathcal{A}$.*
- *Otherwise, there exists a procedure computing* $\inf(\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}))$ *and* $\sup(\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}))$ *in polynomial space in the size of $\mathcal{A}$.*

Proof. If $\mathcal{A}$ has a single clock, then there are $2 * (M + 1)$ regions. Hence, the size of the corner-point abstraction is linear in the size of $\mathcal{A}$. Then, a reward-diverging cycle can be guessed and its ratio can be computed in logarithmic space.

Moreover, given a one-clock timed automaton $\mathcal{A}$ and a value $\mathsf{r} \in [0, 1]$, the problem asking whether there exists a reward-diverging cycle with a ratio smaller than r, is in co-NLOGSPACE, which is equal to NLOGSPACE.

As a consequence, a non-deterministic procedure can guess the minimal (resp. maximal) reward-diverging cycle and compute its ratio and then check that it is optimal, all in logarithmic space.

If $\mathcal{A}$ has several clocks, then the number of regions is exponential in the number of clocks. Hence, the size of the corner-point abstraction is exponential in the size of $\mathcal{A}$. The same reasoning as for one-clock timed automata also applies for timed automata with several clocks. The complexity is thus polynomial. As a consequence of the Theorem of Savitch, there exists a deterministic procedure in polynomial space which computes $\inf(\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}))$ and $\sup(\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}))$. □

5.4 Forgetfulness

Forgetfulness was originally defined in [BA11] using the orbit graph [Pur00]. We choose here to give an alternative notion of forgetfulness based on the corner-point abstraction, which is less succinct, but which is useful for computing frequencies. The forgetfulness is used in the rest of the part to define a large class of timed automata for which we can compute the set of frequencies. More precisely, it permits to detect convergences of clocks in timed automata.

5.4.1 Forgetfulness and aperiodicity

Definition 5.12 (forgetfulness)**.**

- *A cycle C of $\mathcal{A}$ is* forgetful *if* $\mathsf{Proj}(C)$ *is strongly connected in $\mathcal{A}_{cp}$;*
- *A timed automaton is* forgetful *if all its simple cycles are forgetful;*
- *A timed automaton is* strongly forgetful *if all its cycles are forgetful.*

Intuitively, forgetful cycles are cycles where some choices of current delays cannot impact forever on the future delays. These cycles can forget previous delays in their long term behaviors. Figure 5.2 represents a timed automaton, inspired by [CHR02], that is not forgetful. Indeed, the projection of the single cycle of this timed automaton is the subgraph with bold edges in its corner-point represented in Figure 5.8, which is clearly not strongly connected. In the location ℓ_1, clock x converges to 1 along

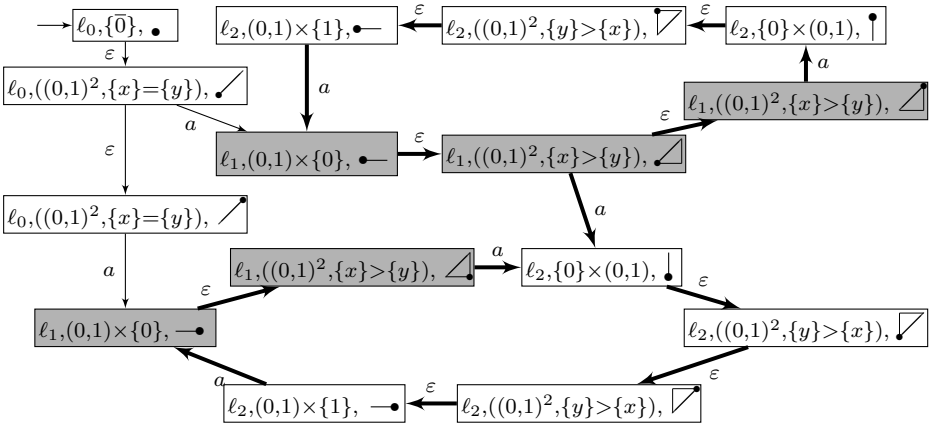


Figure 5.8: Corner-point of the timed automaton from Figure 5.2.

iterations of the cycle. If we visit the state $(\ell_0, \frac{1}{2})$, then state $(\ell_0, \frac{1}{3})$ is no longer reachable. In fact, if from location ℓ_1 an a is read with x close to 0, it becomes impossible to read an a with x close to 1 in the future. More precisely, delays in ℓ_1 are smaller and smaller. From a state $(\ell_1, (\varepsilon_0, 0))$, after a delay ε with $0 < \varepsilon < 1 - \varepsilon_0$, one reads a, then the delay ε' elapsed in ℓ_2 is necessarily $1 - \varepsilon$, thus the state reached reading a is $(\ell_1, (1 - \varepsilon, 0))$ and the next delay ε'' has to satisfy the constraint $\varepsilon > \varepsilon'' > 0$. This is linked with the fact that state $(\ell_0, (0, 1) \times \{0\}, 0^{\{x,y\}})$ is not reachable from state $(\ell_0, (0, 1) \times \{0\}, 0^{\{x,y\}})$ in the corner-point abstraction. On the contrary, in forgetful cycle, all the corners are always reachable from the other ones. It roughly corresponds to the fact that with enough iterations, one can go closer than any corner of the region. One can, in a sense, forget that we have been close to another corner. Note that in Figure 5.8, we did not draw the edges labeled by ε which lead to states from which no discrete transition can be fired in the future.

The strong forgetfulness assumption is stronger than forgetfulness because the condition concerns all the cycles instead of only simple cycles. The concatenation of two forgetful cycles may be not forgetful, even if examples of forgetful timed automata that we built are degenerated as discussed above. They concern cycles which seem to be periodic, visiting alternatively one corner and the other one. The concatenation of such a cycle with itself is then not forgetful, because it visits the same corner at each iteration. We then define the notion of aperiodicity of a forgetful cycle and forgetful aperiodic timed automata. This notion will allow to relax strong forgetfulness in the sequel.

Definition 5.13 (aperiodicity)**.**

- *A forgetful cycle C in a timed automaton is* aperiodic *if for all $k \in \mathbb{N}$, the cycle obtained by the concatenation of k iterations of C is forgetful.*
- *A forgetful timed automaton is* aperiodic *if all its simple cycles are aperiodic;*

Strong forgetfulness trivially implies aperiodicity, whereas forgetfulness does not. Indeed Figure 5.9 represents a timed automaton which is forgetful but periodic (*i.e.* not aperiodic). Its corner-point abstraction illustrates the periodicity, for readability, we only represent the discrete transitions. The cycle formed of two iterations of the simple cycle is not strongly connected, it has two distinct connected components.

The projection of a forgetful cycle C in $\mathcal{A}_{cp}$ is strongly connected, then given any state s of $\mathcal{A}_{cp}$ in $\mathsf{Proj}(C)$, there are some simple cycles containing s. Intuitively, such a cycle corresponds to a

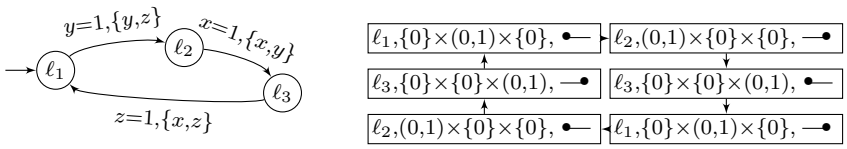


Figure 5.9: A forgetful and periodic timed automaton.

number of iterations of C in $\mathcal{A}$. This is the number of non-consecutive occurrences of states sharing the same location of $\mathcal{A}$ as s. For example, the cycle of the corner-point abstraction on Figure 5.9 corresponds to two iterations of the cycle of the timed automaton. Given a cycle D in $\mathsf{Proj}(C)$, let us note $\mathsf{numb}(D)$ the number of iterations of C corresponding to D. Moreover, we write $\mathsf{SC}(\mathsf{Proj}(C))$ for the set of simple cycles of $\mathsf{Proj}(C)$. Thanks to these notations, we can characterize the aperiodicity of a forgetful cycle by a notion of pseudo aperiodicity of its projection.

Proposition 5.1. *A forgetful cycle C is aperiodic if and only if* $(*)$ $\gcd_{D\in\mathsf{SC}(\mathsf{Proj}(C))}\mathsf{numb}(D)=1$.

Proof of Proposition 5.1. Let us assume that $(*)$ is false, and prove that C is periodic. Let $d\neq 1$ be the greatest common divisor defined in $(*)$. It implies, in particular, that for all states s of $\mathcal{A}_{cp}$ in $\mathsf{Proj}(C)$, there is a cycle (not necessarily simple) in $\mathsf{Proj}(C)$ containing s and corresponding to md iterations of C for $m\in\mathbb{N}$. The goal is to prove that the cycle C^d of $\mathcal{A}$ formed of d iterations of C is not forgetful. Let us fix $s=(\ell,R,\alpha)$ and $s'=(\ell,R,\alpha')$, two states such that an iteration of C allows to go from s to s', that is there is a finite run in $\mathsf{Proj}(C)$ corresponding to a single iteration of C in $\mathcal{A}$ (with the same correspondence as for cycles). If C^d is forgetful, then the projection of C^d is strongly connected and thus there exists a finite run from s' to s in $\mathsf{Proj}(C)$ corresponding to md iterations of C in $\mathcal{A}$ with $m\in\mathbb{N}$. Then, removing the cycles along this finite run, we obtain a finite cycle-free run from s' to s in $\mathsf{Proj}(C)$ corresponding to $m'd$ iterations of C in $\mathcal{A}$ with $m'\leq m$. Hence, there is a simple cycle containing s (and s') and corresponding to $m'd+1$ iterations of C in $\mathcal{A}$, which contradicts that $d\neq 1$.

On the other hand, let us assume that there is a set of pairs (s_i,d_i) such that for all i:

- s_i is a state of $\mathcal{A}_{cp}$ in $\mathsf{Proj}(C)$,
- there is a simple cycle $D_i\in\mathsf{SC}(\mathsf{Proj}(C))$ containing s_i with $\mathsf{numb}(D)=d_i$,
- the greatest common divisor of d_i's is 1.

Then we want to prove that, for all k, we can go from any state $s=(\ell,R,\alpha)$ to any state $s'=(\ell,R,\alpha')$ of $\mathcal{A}_{cp}$ in $\mathsf{Proj}(C)$ with a finite run corresponding to a number of iterations multiple of k. Let us consider a finite run of $\mathcal{A}_{cp}$ in $\mathsf{Proj}(C)$ corresponding to k' iterations which visits all the s_i. Thus, we can add k'' iterations from these s_i such that $k'+k''=k'''k$ for some k''' because the greatest common divisor is 1. Therefore C is aperiodic. □

The characterization $(*)$ of aperiodicity can be effectively checked in the corner-point abstraction. The notion of aperiodicity will be a key for the relaxation of strong forgetfulness (which we do not know how to check) when expressing the set of frequencies in n-clock timed automata.

5.4.2 Comparison with the forgetfulness of [BA11]

Forgetfulness was introduced in [BA11] using the orbit graph abstraction. Roughly, the orbit graph of a cycle looping over a region r in a timed automaton split in regions is the graph whose vertices are the corners of r and there is an edge between two corners if it is possible to go from the source of the edge to it target following the cycle (assuming that guards are closed). It looks like a quotient of the projection in the corner-point abstraction of the cycle.

Let us recall the context of [BA11] to detail the comparison of both notions of forgetful cycle. Timed automata are assumed to be split in regions and regions are open and bounded. The orbit graph is then defined in the following way.

For a closed region $\overline{r}$, let us denote by $V(r) = \{S_1, ..., S_p\}$ its vertices. Any point x in the region is uniquely described by its barycentric coordinates $\lambda_1, \cdots, \lambda_p$, *i.e.* non-negative numbers such that $\Sigma_{i=1}^{p}\lambda_i = 1$; $x = \Sigma_{i=1}^{p}\lambda_i S_i$. Given two regions $\overline{r}$ and $\overline{r'}$, we call *orbit graph* any graph G with vertices $V(r) \sqcup V(r')$ if r and r' are different and $V(r)$ otherwise, and with edges going from $V(r)$ to $V(r')$. Informally, an edge from S to S' means that the clock vector at the vertex S can reach the clock vector at S' along some transition or path. Orbit graphs compose in the natural way: for G_1 on regions $\overline{r_1}$ and $\overline{r'_1}$, and G_2 on regions $\overline{r_2}$ and $\overline{r'_2}$, their product $G = G_1 \cdot G_2$ is defined if $\overline{r'_1} = \overline{r_2}$. In this case, G is an orbit graph on $\overline{r_1}$ and $\overline{r'_2}$. There is an edge from S to S'' in G if and only if there exists S' such that (S, S') and (S', S'') are edges of G_1 and G_2 respectively. Whenever $\overline{r'_1} \neq \overline{r_2}$, we put $G_1 \cdot G_2$ equal to some special (absorbing) element $\mathsf{0}$. The set $\mathcal{G}$ of orbit graphs, augmented with $\mathsf{0}$ and a neutral element $\mathsf{1}$ has a structure of finite monoid.

An orbit graph G can be represented by its adjacency matrix M of size $|V(r)| \times |V(r')|$. Products in the monoid of orbit graphs are easy to compute using matrices: $M(G_1 \cdot G_2) = M(G_1) \otimes M(G_2)$ where the "product" $\otimes$ is defined by

$$(A \otimes B)_{ij} = \max_k \min(A_{ik}, B_{kj}).$$

There exists a natural morphism $\gamma : E^* \to \mathcal{G}$ from paths to orbit graphs defined as follows. For a transition e between r and r', we define the orbit graph $\gamma(e)$ on r and r' with edges $\{(S, S') \in V(r) \times V(r') | \exists t, S \xrightarrow{(e,t)} S'\}$. For a path $\pi = e_1...e_n$, we define $\gamma(\pi) = \gamma(e_1)...\gamma(e_n)$ (it will be called the orbit graph of the path π). For the empty path we have $\gamma(\varepsilon) = \mathsf{1}$, and for any non-consecutive path $\gamma(\pi) = \mathsf{0}$.

For example, the orbit graph of the cycle of the timed automaton from Figure 5.2, considering the location ℓ_1 and the region $((0,1)^2, \{x\} > \{y\})$ is represented in Figure 5.10. Note that its projection in the corner-point abstraction is represented in Figure 5.8.

In [BA11], there are 4 equivalent characterizations of the forgetful cycles. The definition which is easily compared with ours is the completeness of the orbit graph of this cycle.

In fact, by definition of the projection in the corner-point abstraction, the orbit graph of a cycle c on a region r corresponding to the location ℓ can be computed from $\mathsf{Proj}(c)$. The vertices of $\overline{r}$ are the corners of r. c being a cycle, $\gamma(c)$ has only the vertices $V(r)$. Thus, there is an edge from a vertex S of r to a vertex S' of r in $\gamma(c)$ iff there is a path in $\mathsf{Proj}(c)$ from (ℓ, r, S) to (ℓ, r, S') corresponding to an iteration of c. Hence, c is forgetful in our sense if $\gamma(c)$ is strongly-connected which is a weaker condition than $\gamma(c)$ is complete. As a consequence, our notion of forgetfulness is weaker than the BA-forgetfulness.

Then, a cycle is also said aperiodic if all the k-compositions (there is an edge between two vertices in the k-composition of a graph G if there is a path of length k in between them in G) of its orbit graph are strongly-connected. In particular, Proposition 5.1 implies that if a cycle C is forgetful and

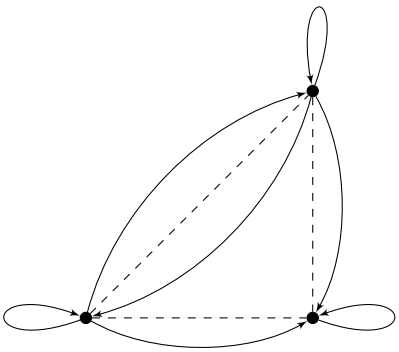


Figure 5.10: An orbit graph of the cycle of the timed automaton from Figure 5.2

aperiodic, then there exists k such that the k-composition of its orbit graph is complete. In other words, the power k of C is BA-forgetful.

In the context of [BA11], one wants to decide whether a timed automaton contains one BA-forgetful cycle, which is done by building the submonoïd of orbit graphs. The existence of a forgetful aperiodic cycle is an equivalent condition. However, we need to work in timed automata with stronger forgetfulness behaviors. Indeed, we are not searching a particular run or cycle, but we are studying the set of the frequencies of all the runs in a timed automaton. Nevertheless, we would like to be able to decide the forgetfulness condition that we are going to use for timed automata. That is the motivation for these finer notions of forgetfulness.

Chapter 6

Frequencies in One-Clock Timed Automata

Introduction

In Chapter 5, we first introduced the notion of frequency of an infinite run as the proportion of time elapsed in accepting locations. Then, we presented the corner-point abstraction, its instrumentation with costs and rewards, and an analogue to frequency in the corner-point abstraction, called ratio.

In this chapter, we show that the infimum and supremum values of frequencies in a given one-clock timed automaton are exactly the infimum and supremum values of ratios in its corner-point abstraction. To do so, we study the links between frequencies in a timed automaton and ratios in its corner-point abstraction. Moreover, we present a way to decide whether these bounds are realizable (*i.e.*, whether they are minimum and maximum respectively) in the timed automaton. A motivation to compute the bounds of the set of the frequencies and to study their realizability is to decide the emptiness problem for languages defined by a threshold on the frequency.

Our restriction to one-clock timed automata is crucial since at several places the techniques employed do not extend to two clocks or more. Some counter-examples with two clocks illustrate this restriction all along the presentation of the intermediate results in this chapter.

The rest of this chapter is structured as follows. We first present relations between runs in timed automata and projections in their corner-point abstraction, and *vice-versa*. Second, we draw the consequences about the possible frequencies by studying, in particular, the realizability of the bounds.

6.1 From $\mathcal{A}$ to $\mathcal{A}_{cp}$

In this section, we first expose how to build two runs in the projection in the corner-point abstraction of any run ϱ having respectively a smaller and a larger ratio than the frequency of ϱ. The construction is intuitive: we simply choose, along the abstraction, the largest possible abstract delays in non-accepting locations and the smallest (resp. largest) delays in accepting locations. In a second phase, we prove that runs of the corner-point abstraction can be mimicked in the timed automaton while preserving the value of the ratio up to any $\varepsilon > 0$. Combining both results, we will derive that the infimum and the maximum of the set of frequencies in a timed automaton are the same as for the set of ratios in its corner-point abstraction.

6.1.1 Contraction and dilatation

We are going to show that given a run ϱ of a timed automaton $\mathcal{A}$, there exists a run in the projection of ϱ, whose ratio is smaller (resp. larger) than the frequency of ϱ.

Proposition 6.1 (From $\mathcal{A}$ to $\mathcal{A}_{cp}$)**.** *For every run ϱ in $\mathcal{A}$, there exist π and π' in $\mathcal{A}_{cp}$ that can effectively be built and belong to $\mathsf{Proj}_{cp}(\varrho)$ such that:*

$$\mathsf{Rat}(\pi) \leq \mathsf{freq}_{\mathcal{A}}(\varrho) \leq \mathsf{Rat}(\pi') and$$

they respectively minimizes and maximizes the ratio among runs in $\mathsf{Proj}_{cp}(\varrho)$.

Such two runs of $\mathcal{A}_{cp}$ can be effectively built from ϱ, through the so-called *contraction* (resp. *dilatation*) operations. Intuitively it consists in minimizing (resp. maximizing) the time elapsed in F-locations. In this section, we first define formally these special projections and then prove that they satisfy the expected property.

Definition of F-dilatation. Let $\varrho = (\ell_0, 0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \cdots$ be a run of $\mathcal{A}$. We note $e_0, e_1, \cdots$ the edges fired along ϱ. We define the F-dilatation (or simply dilatation) of ϱ as the run $\pi = (\ell_0, R_0^1 = \{0\}, \alpha_0^1 = \bullet) \to (\ell_0, R_0^2, \alpha_0^2) \to \cdots \to (\ell_0, R_0^{k_0}, \alpha_0^{k_0}) \to (\ell_1, R_1^1, \alpha_1^1) \to \cdots \to (\ell_1, R_1^{k_1}, \alpha_1^{k_1}) \cdots \in \mathsf{Proj}_{cp}(\varrho)$ in $\mathcal{A}_{cp}$ defined inductively as follows. Assume n transitions of ϱ are reflected in π: π starts with $(\ell_0, R_0^1, \alpha_0^1) \to \cdots \to (\ell_0, R_0^{k_0}, \alpha_0^{k_0}) \to (\ell_1, R_1^1, \alpha_1^1) \to \cdots \to (\ell_n, R_n^1, \alpha_n^1)$ with $v_n \in R_n^1$ and α_n^1 a corner-point of R_n^1.

- if $v_n + \tau_n \leq M$:
  - if $v_n + \tau_n \in R_n^1 = (c, c+1)$, $\alpha_n^1 = \bullet$– and $\ell_n \in F$, then we let time elapse as much as possible and choose in $\mathcal{A}_{cp}$ the portion of path $(\ell_n, R_n^1, \bullet\text{–}) \to (\ell_n, R_n^1, \text{–}\bullet) \to (\ell_{n+1}, R_{n+1}^1, \alpha_{n+1}^1)$ where $(R_{n+1}^1, \alpha_{n+1}^1)$ is the successor pointed region of $(R_n^1, \text{–}\bullet)$ by transition e_n.
  - if $v_n + \tau_n \in R_n^1 = (c, c+1)$, $\alpha_n^1 = \bullet$– and $\ell_n \notin F$, we choose to fire e_n as soon as possible by selecting the following portion of path: $(\ell_n, R_n^1, \bullet\text{–}) \to (\ell_{n+1}, R_{n+1}^1, \alpha_{n+1}^1)$ where $(R_{n+1}^1, \alpha_{n+1}^1)$ is the successor pointed region of $(R_n^1, \bullet\text{–})$ by transition e_n.
  - if $v_n + \tau_n \in R_n^1 = (c, c+1)$ and $\alpha_n^1 = \text{–}\bullet$ is the last corner of R_n^1 (that is the second one), we need to immediately fire e_n in $\mathcal{A}_{cp}$ and thus choose $(\ell_n, R_n^1, \text{–}\bullet) \to (\ell_{n+1}, R_{n+1}^1, \alpha_{n+1}^1)$ where $(R_{n+1}^1, \alpha_{n+1}^1)$ is the successor pointed region of $(R_n^1, \text{–}\bullet)$ by transition e_n.
  - if $v_n + \tau_n \notin R_n^1$ and $\ell_n \notin F$, we fire e_n as soon as possible, that is, we let time elapse until region $R_n^{k_n}$ with $v_n + \tau_n \in R_n^{k_n}$ and its first corner-point $\alpha_n^{k_n}$, and then fire e_n: $(\ell_n, R_n^1, \alpha_n^1) \to \cdots \to (\ell_n, R_n^{k_n}, \alpha_n^{k_n}) \to (\ell_{n+1}, R_{n+1}^1, \alpha_{n+1}^1)$ where $(R_{n+1}^1, \alpha_{n+1}^1)$ is the successor pointed region of $(R_n^{k_n}, \alpha_n^{k_n})$ by transition e_n.
  - if $v_n + \tau_n \notin R_n^1$ and $\ell_n \in F$, we fire e_n as late as possible, that is, we let time elapse until region $R_n^{k_n}$ with $v_n + \tau_n \in R_n^{k_n}$ and its last corner-point $\alpha_n^{k_n}$, and then fire e_n: $(\ell_n, R_n^1, \alpha_n^1) \to \cdots \to (\ell_n, R_n^{k_n}, \alpha_n^{k_n}) \to (\ell_{n+1}, R_{n+1}^1, \alpha_{n+1}^1)$ where $(R_{n+1}^1, \alpha_{n+1}^1)$ is the successor pointed region of $(R_n^{k_n}, \alpha_n^{k_n})$ by transition e_n.
- if $v_n + \tau_n > M$:

- if $R_n^1 \neq (M, +\infty)$, we let time elapse until region $(M, +\infty)$ and add a delay, to respect the definition of the projection in $\mathcal{A}_{cp}$, which depends on ℓ_n and then fire e_n: $(\ell_n, R_n^1, \alpha_n^1) \to \cdots \to (\ell_n, R_n^i, \alpha_n^i) \to (\ell_n, (M, +\infty), \bot)\big(\to (\ell_n, (M, +\infty), \bot)\big)^{\nu_n} \to (\ell_{n+1}, R_{n+1}^1, \alpha_{n+1}^1)$ where $\nu_n = \begin{cases} \lceil v_n + \tau_n \rceil - M & \text{if } \ell_n \in F \\ \lfloor v_n + \tau_n \rfloor - M & \text{if } \ell_n \notin F \end{cases}$ and $(R_{n+1}^1, \alpha_{n+1}^1)$ is the successor pointed region of $((M, +\infty), \bot)$ by transition e_n.
- if $R_n^1 = (M, +\infty)$, respecting the definition of the projection give two possible delays, our choice depends on ℓ_n, then we fire e_n: $(\ell_n, (M, +\infty), \bot)\big(\to (\ell_n, (M, +\infty), \bot)\big)^{\nu_n} \to (\ell_{n+1}, R_{n+1}^1, \alpha_{n+1}^1)$ where $\nu_n = \begin{cases} \lceil v_n + \tau_n \rceil - \nu_{n-1} & \text{if } \ell_n \in F \\ \lfloor v_n + \tau_n \rfloor - \nu_{n-1} & \text{if } \ell_n \notin F \end{cases}$ and $(R_{n+1}^1, \alpha_{n+1}^1)$ is the successor pointed region of $((M, +\infty), \bot)$ by transition e_n.

Similarly, we define the F-contraction of ϱ as its $\bar{F}$-dilatation, *i.e.* the run $\pi \in \mathsf{Proj}_{cp}(\varrho)$ of $\mathcal{A}_{cp}$ which fires transition e_n as soon as possible when $\ell_n \in F$ and as late as possible when $\ell_n \notin F$.

Proof of Proposition 6.1. The proof is based on the following intuitive lemma, whose proof is tedious but not difficult. Given a run $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \cdots$, in the sequel we abusively denote by $\mathsf{freq}_{\mathcal{A}}(\varrho_{|n})$ the quantity given by $(\sum_{i \leq n | \ell_i \in F} \tau_i)/(\sum_{i \leq n} \tau_i)$. In the same spirit, given π a run in $\mathcal{A}_{cp}$, we abusively denote by $\mathsf{Rat}(\pi_{|n})$, the ratio of accumulated costs divided by accumulated rewards for the finite prefix containing $n+1$ discrete transitions.

Lemma C. *Let ϱ be a run of $\mathcal{A}$, and for all $n \in \mathbb{N}$ $\pi_{|n}$ be the dilatation of $\varrho_{|n}$. For all $n \in \mathbb{N}$, if $\mathsf{freq}_{\mathcal{A}}(\varrho_{|n}) = \frac{c_n}{r_n}$ and $\mathsf{Rat}(\pi_{|n}) = \frac{C_n}{R_n}$ then $C_n \geq c_n$ and $(R_n - C_n) \leq (r_n - c_n)$.*

Assuming the latter lemma, it is easy to conclude that $\mathsf{freq}_{\mathcal{A}}(\varrho) \leq \mathsf{Rat}(\pi)$ for π the dilatation of ϱ. Indeed, given $n \in \mathbb{N}$, $R_n - C_n \leq r_n - c_n$ and $c_n > 0$ (the case $c_n = 0$ is straightforward) imply $\frac{R_n - C_n}{c_n} \leq \frac{r_n - c_n}{c_n}$. Moreover, $C_n \geq c_n$. Hence $\frac{R_n - C_n}{C_n} \leq \frac{R_n - C_n}{c_n}$. All together, this yields $\frac{R_n}{C_n} - 1 \leq \frac{r_n}{c_n} - 1$ which is equivalent to $\frac{C_n}{R_n} \geq \frac{c_n}{r_n}$. When n tends to infinity, we obtain $\mathsf{Rat}(\pi) \geq \mathsf{freq}_{\mathcal{A}}(\varrho)$.

Using the fact that the F-contraction of ϱ is the $\bar{F}$-dilatation of ϱ, one obtains $\mathsf{Rat}(\pi') \leq \mathsf{freq}_{\mathcal{A}}(\varrho)$ for π' the contraction of ϱ.

Moreover, the same reasoning can be applied to any run of the projection of ϱ in the corner-point instead of ϱ in $\mathcal{A}$ to prove that the dilatation and the contraction respectively maximizes and minimizes the ratios. □

Let us now prove the Lemma C. The proof is a tedious inspection of cases, the rest of the section starts again on page 138.

Proof of Lemma C. The proof is by induction on n. The base case, for $n = 0$ is trivial, since R_0, C_0, r_0, c_0 are all set to 0 by convention. Note that an initialization at step $n = 1$ would also be possible using cases 1 to 3 in the following cases enumeration.

Let us fix $n \in \mathbb{N}$. Assume now that the conditions of the lemma are satisfied for all $j \leq n$, that is $C_j \geq c_j$ and $(R_j - C_j) \leq (r_j - c_j)$, and let us prove them for $n+1$. Consider the prefix of length $n+1$ of ϱ: $\varrho_{n+1} = (\ell_0, 0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \cdots (\ell_n, v_n) \xrightarrow{\tau_n, a_n} (\ell_{n+1}, v_{n+1})$. We note $e_0, e_1, \ldots$ the edges fired along ϱ. Let us detail a careful inspection of cases, depending on the value of $v_n + \tau_n$ and whether $\ell_n \in F$.



(a) Case 1 ($\mathcal{T}_n > \tau_n$). (b) Case 2 ($\mathcal{T}_n < \tau_n$).

Figure 6.1: Cases 1 and 2.



(a) Case 3 ($\mathcal{T}_n = \tau_n$). (b) Case 4.1 ($\mathcal{T}_n > \tau_n$).

Figure 6.2: Cases 3 and 4.1.

Case 1 $v_n \in \mathbb{N}$, $\ell_n \in F$, and $v_n + \tau_n \notin \mathbb{N}$.
In this case, $\tau_n = \mathcal{T}_n - \tau'$ with $\mathcal{T}_n \in \mathbb{N}$ and $\tau' \in (0,1)$. By definition of the dilatation, π_{n+1} is built from $\pi_{|n}$ by firing $\mathcal{T}_n$ idling transitions weighted $1/1$ in $\mathcal{A}_{cp}$ (possibly interleaved with idling transitions weighted $0/0$) followed by the discrete transition weighted $0/0$ that corresponds to e_n. Thus, $C_{n+1} = C_n + \mathcal{T}_n$, $R_{n+1} = R_n + \mathcal{T}_n$, whereas $c_{n+1} = c_n + \tau_n$ and $r_{n+1} = r_n + \tau_n$. In particular, $C_n \geq c_n$ (induction hypothesis) and $\tau_n < \mathcal{T}_n$ imply $C_{n+1} \geq c_{n+1}$. Moreover, $R_{n+1} - C_{n+1} = R_n - C_n$ and $r_{n+1} - c_{n+1} = r_n - c_n$, and by induction hypothesis $R_n - C_n \leq r_n - c_n$. Hence $R_{n+1} - C_{n+1} \leq r_{n+1} - c_{n+1}$.

Case 2 $v_n \in \mathbb{N}$, $\ell_n \notin F$, and $v_n + \tau_n \notin \mathbb{N}$.
Here, $\tau_n = \mathcal{T}_n + \tau'$ with $\mathcal{T}_n \in \mathbb{N}$ and $\tau' \in (0,1)$. In the dilatation, $\mathcal{T}_n$ transitions weighted $0/1$ will be fired before taking the transition corresponding to e_n. Thus $R_{n+1} = R_n + \mathcal{T}_n$, $C_{n+1} = C_n$, whereas $c_{n+1} = c_n$ and $r_{n+1} = r_n + \mathcal{T}_n + \tau'$. We immediately deduce that $C_{n+1} \geq c_{n+1}$ using the induction hypothesis. Moreover $R_{n+1} - C_{n+1} = R_n + \mathcal{T}_n - C_n \leq r_n - c_n + \mathcal{T}_n < r_n + \mathcal{T}_n + \tau' - c_n = r_{n+1} - c_{n+1}$, where the second step uses the induction hypothesis.

Case 3 $v_n \in \mathbb{N}$, and $v_n + \tau_n \in \mathbb{N}$.
In this case, $\tau_n = \mathcal{T}_n \in \mathbb{N}$ and exactly $\mathcal{T}_n$ transitions with reward 1 will be taken in $\mathcal{A}_{cp}$ before firing the transition that corresponds to e_n. In other words, the costs and rewards are exactly matched in the corner-point abstraction: $C_{n+1} - C_n = c_{n+1} - c_n$ and $R_{n+1} - R_n = r_{n+1} - r_n$. Notice that these equalities hold regardless of whether $\ell_n \in F$. Using the induction hypothesis ($C_n \geq c_n$ and $R_n - c_n \leq r_n - c_n$) we easily conclude: $R_{n+1} - C_{n+1} = R_n - C_n + r_{n+1} - c_{n+1} + c_n - r_n \leq r_{n+1} - c_{n+1}$, and $C_{n+1} = c_{n+1} + C_n - c_n \geq c_{n+1}$.

Case 4 $v_n \notin \mathbb{N}$ and $\ell_n \in F$

Case 4.1 Assume first that the corner in the last state of $\pi_{|n}$ is $\bullet$–. Then letting $\mathcal{T}_n = \lceil v_n + \tau_n - \lfloor v_n \rfloor \rceil$, in the dilatation, $\mathcal{T}_n$ idling transitions weighted $1/1$ will be fired in $\mathcal{A}_{cp}$ before firing the discrete transition corresponding to e_n. Thus $C_{n+1} = C_n + \mathcal{T}_n$, $R_{n+1} = R_n + \mathcal{T}_n$, whereas $c_{n+1} = c_n + \tau_n$ and $r_{n+1} = r_n + \tau_n$. We immediately obtain $C_{n+1} \geq c_{n+1}$

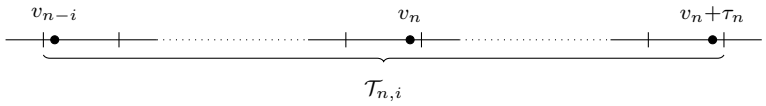


Figure 6.3: Case 4.2

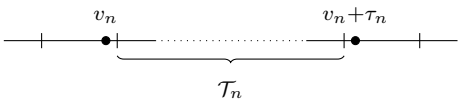


Figure 6.4: Case 5.1 ($\mathcal{T}_n < \tau_n$).

using the induction hypothesis and the fact that $\mathcal{T}_n > \tau_n$. Moreover, $R_{n+1} - C_{n+1} = R_n - C_n \leq r_n - c_n = r_{n+1} - c_{n+1}$.

Note that the picture on Fig. 6.2(b) represents the case $v_n + \tau_n \notin \mathbb{N}$, but the reasoning is valid for $v_n + \tau_n \in \mathbb{N}$ as well.

Case 4.2 Assume now that the corner in the last state of $\pi_{|n}$ is –•. In this situation, we cannot conclude immediately, since $C_{n+1} = C_n + \mathcal{T}_n$ and $c_{n+1} = c_n + \tau_n$, with $\mathcal{T}_n = \lceil v_n + \tau_n - \lceil v_n \rceil \rceil$ is incomparable to τ_n in general. Instead, we need to take into account some previous steps in ϱ and π. Let us consider the least index i such that the corner of the last state in π_{n-i} is not –•. For this index, π_{n-i} ends either with the pointed region $((\lfloor v_{n-i} \rfloor, \lceil v_{n-i} \rceil), \bullet\text{–})$ or with $(\{v_{n-i}\}, \bullet)$. We then consider the suffix of path π_{n+1} after π_{n-i}. Notice that the clock x was not reset along this suffix (since no pointed region of the form $(R, \bullet)$ was reached). For this part, the accumulated reward is $\mathcal{T}_{n,i} = \lceil v_n + \tau_n - \lfloor v_{n-i} \rfloor \rceil$. The corresponding part in ϱ has an accumulated delay $\tau_{n,i} = v_n + \tau_n - v_{n-i} = \sum_{j=n-i}^{n} \tau_j$ (since the clock has not been reset). Note that $\mathcal{T}_{n,i} \geq \tau_{n,i}$.

Let us now discuss the cost accumulated along the suffix of path π_{n+1} after π_{n-i}. By definition of the dilatation, no idling transition can be fired from a state with an F-location along this suffix, otherwise the last state of π_{n-i+1} would not have –• as corner. Thus, the accumulated cost along the suffix of path π_{n+1} after π_{n-i} is equal to $\mathcal{T}_{n,i}$. However, the corresponding part in ϱ has an accumulated cost $c_{n,i}$ smaller than $\tau_{n,i}$ (due to the potential time spent in locations out of F).

The above discussion can be summarized as follows: $R_{n+1} = R_{n-i} + \mathcal{T}_{n,i}$, $C_{n+1} = C_{n-i} + \mathcal{T}_{n,i}$, $r_{n+1} = r_{n-i} + \tau_{n,i}$ and $c_{n+1} = c_{n-i} + c_{n,i}$ with $\mathcal{T}_{n,i} \geq \tau_{n,i} \geq c_{n,i}$. We can thus derive that $C_{n+1} \geq c_{n+1}$, using both the induction hypothesis stating that $C_{n-i} \geq c_{n-i}$ and the fact that $\mathcal{T}_{n,i} \geq c_{n,i}$. It remains to prove that $R_{n+1} - C_{n+1} \leq r_{n+1} - c_{n+1}$. By the above equalities, this is equivalent to proving that $R_{n+i} - C_{n+i} \leq (r_{n+i} - c_{n+i}) + (\tau_{n,i} - c_{n,i})$ which is true by the induction hypothesis stating that $(R_{n+i} - C_{n+i}) \leq (r_{n+i} - c_{n+i})$ and the fact that $\tau_{n,i} \geq c_{n,i}$.

Case 5 $v_n \notin \mathbb{N}$ and $\ell_n \notin F$

Case 5.1 Symmetrically to what precedes, the easy case is when the corner in the last state of $\pi_{|n}$ is –•. Then, letting $\mathcal{T}_n = \lfloor v_n + \tau_n \rfloor - \lceil v_n \rceil < \tau_n$, we can write $R_{n+1} = R_n + \mathcal{T}_n$ and $C_{n+1} = C_n$. Since $c_{n+1} = c_n$ and $r_{n+1} = r_n + \tau_n$, we deduce the desired inequalities.

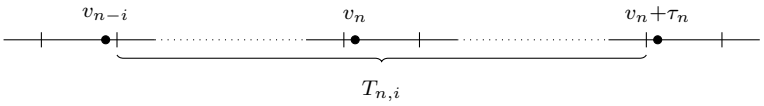


Figure 6.5: Case 5.2.

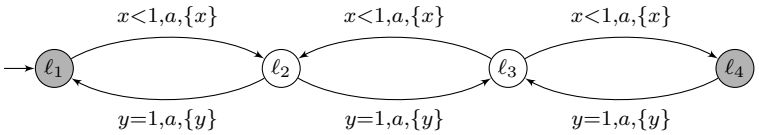


Figure 6.6: A counterexample with two clocks for Proposition 6.1.

Case 5.2 Assume now that the corner in the last state of $\pi_{|n}$ is $\bullet$–. Therefore, the last pointed region in $\pi_{|n}$ is $((\lfloor v_n \rfloor, \lceil v_n \rceil), \bullet\text{–})$, and we let $\mathcal{T}_n = \lfloor v_n + \tau_n - \lfloor v_n \rfloor \rfloor$. By definition of the dilatation, this can only happen if $\ell_{n-1} \notin F$. We then consider the least index i such that the last corner in π_{n-i} is not $\bullet$–. For this index, the last pointed region in π_{n-i} is either $((\lfloor v_{n-i} \rfloor, \lceil v_{n-i} \rceil), \text{–}\bullet)$ or $(\{v_{n-i}\}, \bullet)$. Moreover, all locations ℓ_j for $n - i \leq j \leq n$ are not in F. We define $\mathcal{T}_{n,i} = \lfloor v_n + \tau_n \rfloor - \lceil v_{n-i} \rceil$. Note that $\mathcal{T}_n \leq \sum_{j=n-i}^{n} \tau_j$. Using this notation, $R_{n+1} = R_{n-i} + \mathcal{T}_{n,i}$, and $C_{n+1} = C_{n-j}$. In $\mathcal{A}$, $r_{n+1} = r_{n-j} + \sum_{j=n-i}^{n} \tau_j$ and $c_{n+1} = c_{n-i}$. We trivially derive $C_{n+1} \geq c_{n+1}$ using the analogous induction hypothesis at rank $n-i$. Moreover, $R_{n+1} - C_{n+1} = R_{n-i} + \mathcal{T}_{n,i} - C_{n-i} < R_{n_i} + \sum_{j=n-i}^{n} \tau_j - C_{n-i} \leq r_{n-i} + \sum_{j=n-i}^{n} \tau_j - c_{n-i} = r_{n+1} - c_{n+1}$, using the induction hypothesis at rank $n - i$ in the next to last step.

Let us notice that we ignored the unbounded region through the whole proof. However it can be treated exactly in the same way. Indeed, we can consider the abstract valuations in ϱ instead of the corner-point.

Note that in cases 4.2 and 5.2, the induction relies on other cases (4.1, 5.1, and 1, 2, 3). However, the induction is well-founded since those cases are treated independently. □

6.1.2 Proposition 6.1 does not extend to timed automata with two clocks

Note that the notion of contraction cannot be adapted to the case of timed automata with several clocks, as illustrated by the timed automaton in Fig. 6.6. Indeed, there is a run with frequency $\frac{1}{2}$ whose projections in the corner-point abstraction have all ratio larger than $\frac{2}{3}$. More precisely, consider the run ϱ alternating delays $(\frac{1}{2} + \frac{1}{n})$ and $1 - (\frac{1}{2} + \frac{1}{n})$ for $n \in \mathbb{N}$, and switching between the left-most cycle $(\ell_1 - \ell_2 - \ell_1)$ and the right-most cycle $(\ell_3 - \ell_4 - \ell_3)$ following the rules: in round k, take 2^{2k} times the cycle $\ell_1 - \ell_2 - \ell_1$, then switch to ℓ_3 and take 2^{2k+1} times the cycle $\ell_3 - \ell_4 - \ell_3$ and return back to ℓ_1 and continue with round $k + 1$.

In fact, delays in ℓ_1 and ℓ_3 need to be smaller and smaller. A projection of ϱ thus ends either with rewards 1 in ℓ_1 and ℓ_3 and 0 otherwise, or with rewards 1 in ℓ_2 and ℓ_4 and 0 otherwise. If one chooses one time to put a reward 1 in ℓ_2 or ℓ_4, one never have another possibility to make another choice. This is a convergence phenomenon which requires at least two clocks..

Run ϱ cannot have any contraction since its frequency is $\frac{1}{2}$, whereas all its projections in the corner-point abstraction have ratio larger than $\frac{2}{3}$. Indeed, consider $(h_k)_{k\geq 1}$ the sequence of the proportions of time elapsed in accepting locations when switching from the cycle $\ell_3-\ell_4-\ell_3$ to $\ell_1-\ell_2-\ell_1$ when rewards 1 are elapsed in locations ℓ_2 and ℓ_4:

$$h_k = \frac{\sum_{i=0}^{k} 2^{2i+1}}{k+\sum_{i=0}^{k} 2^{2i}+2^{2i+1}} = \frac{2\sum_{i=0}^{k} 4^i}{k+\sum_{i=0}^{2k+1} 2^i} = \frac{2\frac{4^{k+1}-1}{3}}{k+2^{2k+2}-1} = \frac{2*2^{2k+2}-2}{3*2^{2k+2}+3k-3} = \frac{2-\frac{2}{2^{2k+2}}}{3+\frac{3k-3}{2^{2k+2}}}.$$

This sequence converges to $\frac{2}{3}$. The same computation can be done for the case where rewards 1 are elapsed in locations ℓ_1 and ℓ_3 considering the other switch of cycles. Note that this example uses the non-convergence of the ratio together with the choice of the lim sup for the definition. It is not clear that such an example exists for the dilatation.

6.2 From $\mathcal{A}_{cp}$ to $\mathcal{A}$

We now want to know when and how ratios in $\mathcal{A}_{cp}$ can be lifted to frequencies in $\mathcal{A}$. To that aim we distinguish between reward-diverging and reward-converging runs.

6.2.1 Reward-diverging case

Given a reward-diverging run π in the corner-point abstraction, one can lift it to a non-Zeno run having the same frequency as its ratio.

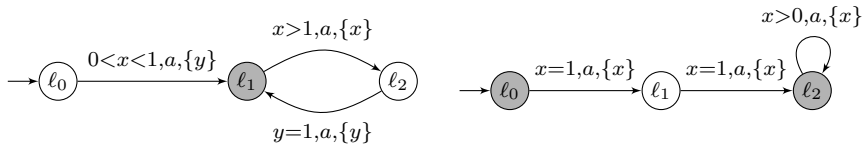
Proposition 6.2 (From $\mathcal{A}_{cp}$ to $\mathcal{A}$, reward-diverging case)**.** *For every reward-diverging run π in $\mathcal{A}_{cp}$, there exists a non-Zeno run ϱ in $\mathcal{A}$ such that $\pi \in \mathsf{Proj}_{cp}(\varrho)$ and $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$.*

The key ingredient is that given a reward-diverging run π in $\mathcal{A}_{cp}$, for every $\varepsilon > 0$, one can build a non-Zeno run ϱ_ε of $\mathcal{A}$ with the following strong property: for all $n \in \mathbb{N}$, the valuation $\varrho_\epsilon[n]$ is $\frac{\epsilon}{2^n}$-close to the abstract valuation $\pi[n]$ (defined on page 122). The accumulated reward along π diverges, hence $\mathsf{freq}_{\mathcal{A}}(\varrho_\varepsilon)$ is equal to $\mathsf{Rat}(\pi)$.

Proof. Given $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0,a_0} (\ell_1, v_1) \xrightarrow{\tau_1,a_1} (\ell_2, v_2) \cdots$ a run and $n \in \mathbb{N}$, we denote by $\varrho[n]$ the valuation $v_n + \tau_n$. Similarly, if π belongs to $\mathsf{Proj}_{cp}(\varrho)$, we consider the states of π which correspond with a state of ϱ (those which are just before a discrete transition) and we note $\pi[n]$ the valuation of the corner of the n-th state if the region is bounded. Otherwise, $\pi[n]$ is the sum of all the rewards since the last region $\{0\}$. Proposition 6.2 relies on the following lemma:

Lemma D. *For every reward-diverging run π in $\mathcal{A}_{cp}$, there exists a run ϱ of $\mathcal{A}$ such that, for all $n \in \mathbb{N}$, $|\pi[n] - \varrho[n]| \leq \frac{1}{2^n}$.*

Proof of Lemma D. We show that given a reward-diverging run π in $\mathcal{A}_{cp}$, we can build a run ϱ such that $\pi \in \mathsf{Proj}_{cp}(\varrho)$ and the $\varrho[i]$ are as close as we want of the $\pi[i]$. More precisely, we show that we can choose suitable delays. In the case where $\pi[i]$ is different than $\pi[i+1]$, the choice of the delay allows to be as close as wanted of $\pi[i+1]$. If $\pi[i]$ and $\pi[i+1]$ are equal but an upper bound of a region, we can move nearer to $\pi[i+1] = \pi[i]$ by the new delay. If the region is unbounded, and $\pi[i+1]$ larger than the maximal constant, it is again a good case. The only difficulty is the case where the new delay force us to move further than $\pi[i+1] = \pi[i]$. The solution is to consider globally the sequence of the delays in the same corner together with the delay leading to it. Thanks to the non-zenoness, this sequence is necessarily finite. Therefore, we can effectively choose suitable delays to respect the condition at the end of the sequence and thus all along the sequence. Note this lemma is a simpler version of the Lemma 3 in [BBL08]. □



(a) A counterexample with two clocks. (b) Zeno case.

Figure 6.7: Counterexamples to extensions of Proposition 6.2.

Let us consider apart the cases where $\mathsf{Rat}(\pi) = 0$ and $\mathsf{Rat}(\pi) > 0$.

Assume first that π is a reward-diverging run in $\mathcal{A}_{cp}$ with $\mathsf{Rat}(\pi) = 0$. Given $\varepsilon > 0$, let ϱ be a run of $\mathcal{A}$ such that, for all $n \in \mathbb{N}$, $|\pi[n] - \varrho[n]| < \frac{\varepsilon}{2^n}$. If C_n and R_n are the accumulated costs and rewards in the first n steps of π inz $\mathcal{A}_{cp}$, then $\mathsf{Rat}(\pi) = \limsup_{n\to+\infty} \frac{C_n}{R_n}$ and $\mathsf{freq}_{\mathcal{A}}(\varrho) = \limsup_{n\to+\infty} \frac{C_n+\sum_{i\leq n} \alpha_i\varepsilon/2^i}{R_n+\sum_{i\leq n} \beta_i\varepsilon/2^i}$ where for every i, $\alpha_i \in \{-1,0,1\}$ and $\beta_i \in \{-1,1\}$. Hence $\mathsf{freq}_{\mathcal{A}}(\varrho) \leq \limsup_{n\to+\infty} \frac{C_n+\varepsilon}{R_n-\varepsilon}$ (because $R_n > \varepsilon$ for n large enough). Since $\lim_{n\to+\infty} \frac{C_n}{R_n} = 0$ and $\lim_{n\to+\infty} R_n = +\infty$, we deduce $\limsup_{n\to+\infty} \frac{C_n+\varepsilon}{R_n-\varepsilon} = 0$ which means $\mathsf{freq}_{\mathcal{A}}(\varrho) = 0 = \mathsf{Rat}(\pi)$.

Assume now that π is a reward-diverging run in $\mathcal{A}_{cp}$ with $\mathsf{Rat}(\pi) > 0$. Using the same notations as in the previous case, $|\frac{C_n}{R_n} - \frac{C_n+\sum_{i\leq n} \alpha_i\varepsilon/2^i}{R_n+\sum_{i\leq n} \beta_i\varepsilon/2^i}| \leq \frac{C_n\varepsilon+R_n\varepsilon}{R_n(R_n-\varepsilon)}$. The latter term tends to 0 as n tends to infinity. As a consequence $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$. □

6.2.2 Proposition 6.2 extends neither to timed automata with two clocks, nor to Zeno runs.

The restriction to one-clock timed automata is crucial in Proposition 6.2. Indeed, consider the two-clocks timed automaton depicted in Fig. 6.7(a). In its corner-point abstraction there exists a reward-diverging run π with $\mathsf{Rat}(\pi) = 0$, however every run ϱ satisfies $\mathsf{freq}_{\mathcal{A}}(\varrho) > 0$. More precisely, we exhibit here a reward-diverging run π in $\mathcal{A}_{cp}$ of ratio zero and explain why every run ϱ in $\mathcal{A}$ has a positive frequency. First, π consists (omitting idling transitions weighted $0/0$) of the following sequence of transitions :

$$(\ell_0,\{0\}^2,\bullet) \xrightarrow{\varepsilon,0/1} (\ell_0,((0,1)^2,\{x\}=\{y\}),\nearrow) \xrightarrow{a,0/0}$$
$$\Big((\ell_1,((1,2)\times(0,1),\{x\}<\{y\}),\swarrow) \xrightarrow{a,0/0} (\ell_2,\{0\}\times(0,1),\downarrow) \xrightarrow{\varepsilon,0/1} (\ell_2,(0,1)\times\{0\},\ \rightarrow\bullet) \xrightarrow{a,0/0}\Big)^{\omega}.$$

The ratio of π is thus zero because the accumulated cost of π is zero whereas the reward diverges. On the other hand, let us consider a run ϱ of $\mathcal{A}$ and prove that its frequency is positive. Indeed, ϱ reads necessarily a word of the form $(1-\tau_0,a).\big((\tau_i,a).(1-\tau_i,a)\big)_{1\leq i}$ where $\tau_0 \in (0,1)$ and $\tau_{i+1} > \tau_i$ for all $0 \leq i$. The frequency of $F = \{\ell_1\}$ in ϱ is thus given by:

$$\mathsf{freq}_{\mathcal{A}}(\varrho) = \limsup_{n\to+\infty} \frac{\sum_{i\leq n} \tau_i}{\sum_{i\leq n} 1} > \limsup_{n\to+\infty} \frac{\sum_{i\leq n} \tau_0}{n}.$$

Hence, $\mathsf{freq}_{\mathcal{A}}(\varrho) > \tau_0 > 0$.

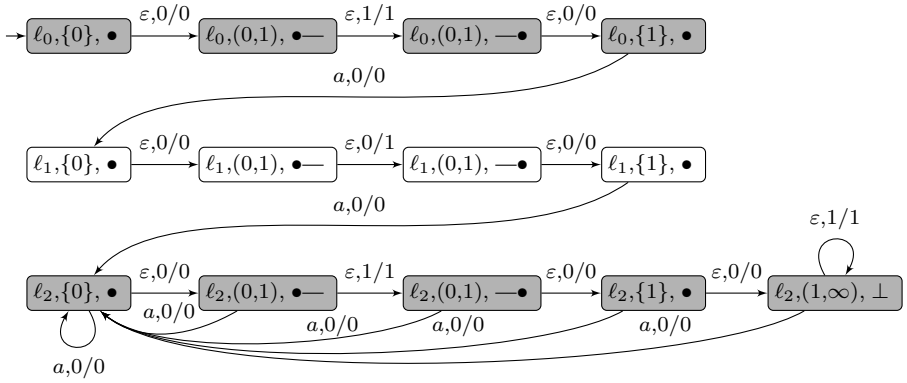


Figure 6.8: The corner-point abstraction $\mathcal{A}_{cp}$ of $\mathcal{A}$ represented Figure 6.7(b).

6.2.3 Proposition 6.2 does not extend to reward-converging runs

An equivalent to Proposition 6.2 for reward-converging runs (even in the one-clock case!) is hopeless. The timed automaton $\mathcal{A}$ depicted in Fig. 6.7(b), where $F = \{\ell_0, \ell_2\}$ is a counterexample. Indeed, in $\mathcal{A}_{cp}$ the reward-converging run π iterating infinitely the loop over state $\ell_2, \{0\},\ \bullet$ has ratio $\frac{1}{2}$, whereas all runs in $\mathcal{A}$ have frequency larger than $\frac{1}{2}$.

6.2.4 Reward-converging case

Even if Proposition 6.2 does not apply to reward-converging runs, a similar construction can be performed. Sometimes, one cannot lift a reward 1 to a delay 1 in $\mathcal{A}$ but only to a delay strictly smaller than 1. Those small imprecisions in the mimicking are negligible when time diverges but impact on the frequency when time converges.

Proposition 6.3 (From $\mathcal{A}_{cp}$ to $\mathcal{A}$, reward-converging case)**.** *For every reward-converging run π in $\mathcal{A}_{cp}$, if $\mathsf{Rat}(\pi) > 0$, then for every $\varepsilon > 0$, there exists a Zeno run ϱ_ε in $\mathcal{A}$ such that $\pi \in \mathsf{Proj}_{cp}(\varrho_\varepsilon)$ and $|\mathsf{freq}_{\mathcal{A}}(\varrho_\varepsilon) - \mathsf{Rat}(\pi)| < \varepsilon$.*

A construction similar to the one used in the proof of Proposition 6.2 is performed. Note however that the result is slightly weaker, since in the reward-converging case, one cannot ignore imprecisions forced *e.g.*, by the prohibition of the zero delays, or by strict inequalities in guards.

Proof. Proposition 6.3 uses the following lemma:

Lemma E. *For every reward-converging run π in $\mathcal{A}_{cp}$, for every $\varepsilon > 0$, there exists a Zeno run ϱ_ε in $\mathcal{A}$ such that $\pi \in \mathsf{Proj}_{cp}(\varrho_\varepsilon)$ and for all $n \in \mathbb{N}$, $|\pi[n](x) - \varrho_\varepsilon[n](x)| < \varepsilon$.*

Proof of Lemma E. Let π be a reward-converging run in $\mathcal{A}_{cp}$, and $\varepsilon \in (0, 1)$. As π is reward-converging, it ends with transitions weighted $0/0$. Let π' be its longest prefix not ending with a transition weighted $0/0$. With π', one can associate a finite run ϱ' of $\mathcal{A}$, as we did for reward-diverging runs (see proof of Lemma D): for all indices i smaller than the length of π', $|\pi'[i](x) - \varrho'[i](x)| < \frac{\varepsilon}{2^i}$. For the suffix of π, composed only of transitions weighted $0/0$, we define a corresponding run in $\mathcal{A}$ with total duration less than ε. This can, *e.g.*, be achieved by taking successive delays of $\frac{\varepsilon}{2^k}$ for $k \geq 1$. Concatenating ϱ' and the run defined above yields a run ϱ^* in $\mathcal{A}$ always $2 * \varepsilon$-close to π (*i.e.* for all $n \in \mathbb{N}$, $|\pi[n](x) - \varrho^*[n](x)| < 2 * \varepsilon$). □

Let us assume that $\mathsf{Rat}(\pi) > 0$. Let n_π be the length of the smallest prefix of π such that there is no transition with non-zero reward after. Thanks to the convergence of the accumulated rewards of π, n_π is necessarily finite. Given $\varepsilon > 0$, the run $\varrho_{\varepsilon'}$ given by the Lemma E with $\varepsilon' = \frac{\varepsilon}{n_\pi+1}$ satisfies the desired property. □

Remark 6.1. *Note that if π is a contraction, then π is the contraction of ϱ_ε defined in the proof of Lemma E.*

Remark 6.2. *If π is of ratio 0, only three cases are possible because zero delays are forbidden:*

- *π visits only accepting locations and the reward of π is 0, then if π is in the projection of ϱ, $\mathsf{freq}_{\mathcal{A}}(\varrho) = 1$,*
- *π visits only non-accepting locations, the frequency of each run ϱ whose projection contains π, is 0*
- *otherwise, neither 1 nor 0 can be the frequency of a run ϱ having π in its projection.*

6.3 Set of frequencies in $\mathcal{A}$

In this section, we use the strong relation between frequencies of runs in $\mathcal{A}$ and ratios of runs in $\mathcal{A}_{cp}$ established in the previous subsection to establish key properties of the set of frequencies. A first consequence of Section 6.1 is that the set of frequencies in a one-clock timed automaton has the same lower and upper bounds as the set of ratios in its corner-point abstraction. Yet these bounds might not be sufficient. For example, to decide emptiness of languages defined with a constraint $\geq \lambda$ over the frequencies, we need to decide the realizability of these bounds.

Theorem 6.1. *The bounds $\inf(\mathsf{Freq}(\mathcal{A}))$ and $\sup(\mathsf{Freq}(\mathcal{A}))$ can be computed and their realizability is decidable. Moreover, both can be done by a non-deterministic procedure in logarithmic space.*

This theorem is based on Proposition 6.4 and Lemma 6.1 dealing respectively with the set of non-Zeno and Zeno runs in $\mathcal{A}$. This section is naturally structured. The next subsection is devoted to the proof that the set of frequencies of the non-Zeno runs in a one-clock timed automaton is exactly the set of ratios of the reward-diverging runs in its corner-point abstraction. Then, Subsection 6.3.2 presents technical lemmas for the decidability of the realizability of the bounds by some Zeno runs. Last, Theorem 6.1 is proved in Subsection 6.3.3.

6.3.1 Set of frequencies of non-Zeno runs in $\mathcal{A}$

Proposition 6.4 (non-Zeno case)**.** $\mathsf{Freq}_{nZ}(\mathcal{A}) = \mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$

Proof. The proposition is based on Lemma B and Theorem 5.1, proved in the preliminary Section 5.3.3. Let $\{C_1, \cdots, C_k\}$ be the set of reachable SCCs of $\mathcal{A}_{cp}$. Recall that Lemma B establishes that the set of ratios of reward-diverging runs of $\mathcal{A}_{cp}$ ending in an SCC C_i is the interval $[m_i, M_i]$, where $m_i = \min(\mathcal{R}_i)$ and $M_i = \max(\mathcal{R}_i)$ and $\mathcal{R}_i$ denotes the set of ratios of reward-diverging simple cycles in C_i. Then Theorem 5.1 gives the general expression $\cup_{1\leq i\leq k}[m_i, M_i]$ for the set of ratios for the reward-diverging runs in $\mathcal{A}_{cp}$. By Proposition 6.2, we know that $\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}) \subseteq \mathsf{Freq}_{nZ}(\mathcal{A})$. Moreover, thanks to the Proposition 6.1 and the convexity of the intervals $[m_i, M_i]$, we can show the other inclusion $\mathsf{Freq}_{nZ}(\mathcal{A}) \subseteq \cup_{1\leq i\leq k}[m_i, M_i] = \mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$ as follows. Let ϱ be a non-Zeno run in $\mathcal{A}$. We distinguish between two cases:

- if the contraction and the dilatation of ϱ are both reward-diverging, then either the clock is reset infinitely often along ϱ, or from some point on, the value of the clock along ϱ lies in the unbounded region forever. In the first case, there is some state of the form $(\ell, \{0\}, \bullet)$ in $\mathcal{A}_{cp}$ which is visited infinitely often by both the contraction and the dilatation. In the second case, from some point on, they will follow the same transitions between states of the form $(\ell, \perp, \alpha_\perp)$ (within the unbounded region). In both cases, the contraction and the dilatation both end up in the same SCC, say C_i. Their ratios, and the frequency of ϱ (thanks to Proposition 6.1 and Lemma B) thus lie in the interval $[m_i, M_i]$.

- if the contraction (resp. dilatation) of ϱ is reward-converging, the frequency of ϱ is 1 (resp. 0). Indeed, if the contraction (resp. dilatation) of ϱ is reward-converging, then the accumulated delay in non-accepting (resp. accepting) locations is bounded in ϱ by the accumulated reward in non-accepting (resp. accepting) locations in the contraction (resp. dilatation). In this case, the dilatation (resp. contraction) is reward-diverging and of ratio 1 (resp. 0), therefore $\mathsf{freq}_{\mathcal{A}}(\varrho) \in \mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$.

As a consequence, the set $\mathsf{Freq}_{nZ}(\mathcal{A})$ of frequencies of non-Zeno runs of $\mathcal{A}$ is equal to the set $\cup_{1\leq i\leq k}[m_i, M_i]$ of ratios of the reward-diverging runs of $\mathcal{A}_{cp}$. □

6.3.2 Realizability of bounds by Zeno runs in $\mathcal{A}$

In this section, we deal with reward-converging runs. To do so, we propose a procedure to decide whether a reward-converging contraction (resp. dilatation) can be lifted to a Zeno run in the timed automaton with frequency equal to the ratio. We prove a lemma for contraction useful for the lower bounds, but of course, a similar lemma for dilatation can be similarly proved.

Lemma 6.1 (Zeno case)**.** *A reward-converging run π in $\mathcal{A}_{cp}$ is a contraction such that there exists a Zeno run ϱ whose contraction is π and with $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$ if and only if π satisfies the following conditions :*

- *from each state of π of the form $(\ell, (i, i+1), \bullet\text{–})$ where $\ell \notin F$, π follows an idling transition to $(\ell, (i, i+1), \text{–}\bullet)$;*

- *after each move $(\ell, (i, i+1), \bullet\text{–}) \xrightarrow{1/1} (\ell, (i, i+1), \text{–}\bullet)$ where $\ell \in F$, π reaches $(\ell, \{i+1\}, \bullet)$ by an idling transition.*

- *If $\mathsf{Rat}(\pi) = 0$, then there are only non-accepting locations along π.*

- *If $0 < \mathsf{Rat}(\pi) < 1$, then*

  - *discrete transitions with resets go out of punctual regions or of the unbounded region; and discrete transitions going from F-locations to $\overline{F}$-locations or the opposite, has to be fired from a punctual region, or from the unbounded region after a positive reward;*
  - *π ends up in non-accepting locations with a constant pointed region of the form $((k, k+1), \text{–}\bullet)$ with k an integer;*

Every fragment of π between reset transitions can be considered independently, since imprecisions cannot be neglected in Zeno runs: even the smallest deviation (such as a delay ε in $\mathcal{A}$ instead of a cost 0 in $\mathcal{A}_{cp}$) will introduce a difference between the ratio and the frequency. A careful inspection of cases allows one to establish the result stated in the lemma.

Proof. Let π be a reward-converging lasso run of $\mathcal{A}_{cp}$. Run π is a contraction if and only if it satisfies the following conditions:

- from each state of π of the form $(\ell, (i, i+1), \bullet\text{–})$ where $\ell \notin F$, π follows an idling transition to $(\ell, (i, i+1), \text{–}\bullet)$;
- after each move $(\ell, (i, i+1), \bullet\text{–}) \xrightarrow{1/1} (\ell, (i, i+1), \text{–}\bullet)$ where $\ell \in F$, π reaches $(\ell, \{i+1\}, \bullet)$ by an idling transition.

This is straightforward from the definition of contraction.

Then, we now study how to decide whether π is a contraction such that there exists a run ϱ whose contraction is π and with $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$. We first consider two simple cases:

- If $\mathsf{Rat}(\pi) = 0$, then there exists a Zeno run ϱ in $\mathcal{A}$ whose contraction is π, such that $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi) = 0$ if and only if there are only non-accepting locations along π. Indeed if there were at least one accepting location, because of the zenoness of ϱ and the positivity of all delays, $\mathsf{freq}_{\mathcal{A}}(\varrho)$ would be positive.
- If $\mathsf{Rat}(\pi) = 1$, let π be the contraction of some run ϱ, then $\mathsf{freq}_{\mathcal{A}}(\varrho) \geq \mathsf{Rat}(\pi)$ hence by definition of the contraction $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$.

We now assume that $0 < \mathsf{Rat}(\pi) < 1$ and π is a contraction. For any run ϱ such that π is the contraction of ϱ, $\mathsf{Rat}(\pi) \leq \mathsf{freq}_{\mathcal{A}}(\varrho)$. Thus in order to get the equality we will have to minimize delays spent in F-locations when building ϱ.

Note that resets of the clock are not reflected in the corner-point abstraction, but could easily be. Indeed, each discrete transition of the corner-point abstraction corresponds to an edge in $\mathcal{A}$ and thus to a set of reset, moreover, states of the corner-point abstraction contain a region which allow to guess which clocks have been reset. Therefore, in the sequel, we abusively speak of resets in π. In the rest of the proof, we will work independently on the reset-free parts of π, let us shortly argue why this reasoning holds in this context. Let ϱ be a path containing finitely many resets and such that $\varrho = \varrho^1 \xrightarrow{a_1} \varrho^2 \xrightarrow{a_2} \cdots \varrho^n$ where all the ϱ^i's are reset-free and the a_i's are reset edges. Let π be the contraction of ϱ, let us notice that π can be written as $\pi^1 \xrightarrow{a_1} \pi^2 \xrightarrow{a_2} \cdots \pi^n$ where π^i corresponds to the contraction of ϱ^i (for $1 \leq i \leq n$). By definition of the contraction, we know that $\mathsf{Rat}(\pi^i) \leq \mathsf{freq}_{\mathcal{A}}(\varrho^i)$ for each $1 \leq i \leq n$. In particular, if there exists i such that $\mathsf{Rat}(\pi^i) \neq \mathsf{freq}_{\mathcal{A}}(\varrho^i)$, it is necessarily the case that $\mathsf{Rat}(\pi^i) < \mathsf{freq}_{\mathcal{A}}(\varrho^i)$. In this situation, it is clearly impossible to obtain $\mathsf{Rat}(\pi) = \mathsf{freq}_{\mathcal{A}}(\varrho)$.

Necessary conditions Now, we are going to prove that if π is a contraction such that there exists a run ϱ whose contraction is π and with $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$, then the following conditions are satisfied:

1. discrete transitions with resets go out of punctual regions or of the unbounded region; and discrete transitions going from F-locations to $\overline{F}$-locations or the opposite, has to be fired from a punctual region, or from the unbounded region after a positive reward;
2. π ends up in only non-accepting locations and stays in a pointed region of the form $((k, k+1), \text{–}\bullet)$.

Let us assume that π is a contraction such that there exists a run ϱ whose contraction is π and with $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$. Run π is reward-converging, then there is ultimately only 0 rewards.

1. Let us first prove that there is a suffix of π which has only non-accepting locations and stays in a pointed region of the form $((k, k+1), -\bullet)$. Because π is reward-converging, ultimately all rewards are 0. Hence, only a finite prefix of π contributes to the ratio. As π is a contraction, after this prefix, there is no state of the form $(\ell, (k, k+1), \bullet-)$ with $\ell \in \overline{F}$ otherwise there would immediately be a transition of reward 1. Thus, there is no state of the form $(\ell, \{k\}, \bullet)$ with $\ell \in \overline{F}$. Then, if there are resets in the suffix, then it contains only states with locations of F and stays in states of the forms $(\ell, \{0\}, \bullet)$ and $(\ell, (0, 1), \bullet-)$. In this case, since delays are positive in $\mathcal{A}$, the equality $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$ cannot hold. As a consequence, π ends up in states of the form $(\ell, (k, k+1), -\bullet)$ with k a fixed integer, and since π is a contraction, the location does not belong to F.

2. Let us now prove that the prefix pref, before the suffix in states of the form $(\ell, (k, k+1), -\bullet)$, is such that discrete transitions with resets go out of punctual regions or of the unbounded region; and discrete transitions going from F-locations to $\overline{F}$-locations or the opposite, has to be fired from a punctual region, or from the unbounded region after a positive reward. These conditions are necessary and sufficient for pref to not prevent that $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$. The proof is based on the disjunction of cases of the proof of Lemma C. In the disjunction, we have seen that for every F-fragment (resp. $\bar{F}$-fragment) of π, the ratio is smaller or equal to the frequency of the corresponding fragment in ϱ. Furthermore, the equality holds only for the case 3 and the cases 4.2 and 5.2 whether v_{n-i} and $v_n + \tau_n$ are integers. Then every sub-fragment (F-fragment or $\bar{F}$-fragment) of a finite fragment (separated by resets) has to correspond to one of these cases. Case 3 is when two consecutive discrete transitions are done in punctual regions. Case 4.2 with v_{n-i} and $v_n + \tau_n$ integers correspond to the case where several discrete transitions can be fired without resets between two punctual regions from an accepting location, then the accumulated rewards and delays are the same. As a consequence, if all the locations are accepting the frequency evolve in the same way as the ratio of π. Finally, the case 5.2 is the same as the case 4.2 but from a non-accepting location.

   The combination of these conditions is equivalent to the two above conditions if regions are bounded. As explained in the proof of Lemma C, the unbounded region can be treated in the same way as bounded ones considering abstract valuations instead of corners. The single subtlety is that the first occurrence of the unbounded region is treated as a pointed region of the form $(R, \bullet-)$ because the delay in $\mathcal{A}$ has to be positive. Then, it can be treated as a pointed region of the form $(R, \bullet)$. We thus obtain the above conditions.

The conditions are sufficient If pref satisfies the conditions for π to be a contraction and pref and the corresponding suffix satisfy the above conditions, then π is a contraction and there exists a run ϱ whose contraction is π and such that $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$. Let pref' be the greatest prefix of pref ending either with a reset, or going from F to $\overline{F}$. We saw that the conditions over pref are sufficient for the existence of ϱ such that the prefix p of ϱ corresponding to pref' is such that $\mathsf{freq}_{\mathcal{A}}(\mathsf{p}) = \mathsf{Rat}(\mathsf{pref}')$. Let us prove that the conditions over the end of π are sufficient. By construction of pref', the suffix that we consider start in a state of the form $(\ell, \{k'\}, \bullet)$ with k' an integer smaller than k, $\ell \in \overline{F}$, and there is no reset in the sequel. Then, the end of ϱ can be simply built by choosing delays whose sum tends to the corresponding reward $k - k'$ in π. □

As a straightforward consequence, lasso runs can be treated in linear time.

Corollary 6.1. *Given a reward-converging lasso run π in $\mathcal{A}_{cp}$, whose cycle-free part is* pref *of length n and whose cycle is* c *of length n', it is decidable in $O(n+n')$ operations, whether there exists a Zeno run ϱ such that π is the contraction of ϱ and $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$.*

A similar lemma can be proved for dilatations, using the fact that the F-dilatation is the $\overline{F}$-contraction:

Lemma 6.2 (Zeno case)**.** *A reward-converging run π in $\mathcal{A}_{cp}$ is a dilatation such that there exists a Zeno run ϱ whose dilatation is π and with $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$ if and only if π satisfies the following conditions :*

- *from each state of π of the form $(\ell, (i, i+1), \bullet\text{–})$ where $\ell \in F$, π follows an idling transition to $(\ell, (i, i+1), \text{–}\bullet)$;*
- *after each move $(\ell, (i, i+1), \bullet\text{–}) \xrightarrow{1/1} (\ell, (i, i+1), \text{–}\bullet)$ where $\ell \notin F$, π reaches $(\ell, \{i+1\}, \bullet)$ by an idling transition.*
- *If $\mathsf{Rat}(\pi) = 1$, then there are only accepting locations along π.*
- *If $0 < \mathsf{Rat}(\pi) < 1$, then*
  - *discrete transitions with resets go out of punctual regions or of the unbounded region; and discrete transitions going from F-locations to $\overline{F}$-locations or the opposite, has to be fired from a punctual region, or from the unbounded region after a positive reward;*
  - *π ends up in accepting locations with a constant pointed region of the form $((k, k+1), \text{–}\bullet)$ with k an integer;*

In the same way, we also obtain the corresponding corollary.

Corollary 6.2. *Given a reward-converging lasso run π in $\mathcal{A}_{cp}$, whose cycle-free part is* pref *of length n and whose cycle is* c *of length n', it is decidable in $O(n+n')$ operations, whether there exists a Zeno run ϱ such that π is the dilatation of ϱ and $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$.*

6.3.3 Proof of Theorem 6.1

Using Propositions 6.1 and 6.4, and Lemmas 6.1 and 6.2, let us briefly explain how we derive Theorem 6.1. For each SCC C of the corner-point abstraction $\mathcal{A}_{cp}$, the bounds of the set of frequencies of runs whose contraction ends up in C can be computed thanks to the above results. Furthermore we can also decide whether these bounds can be obtained by a real run in $\mathcal{A}$. The result for the global automaton follows.

Proof of Theorem 6.1. Let us detail the steps to compute $\inf(\mathsf{Freq}(\mathcal{A}))$ and to decide whether it is realized by a run.

- Computing r_{cycle}, the minimal ratio of a simple reward diverging cycle in $\mathcal{A}_{cp}$;
- Computing r_{prefix}, the minimal ratio of a cycle-free prefix ending in a state of a reward-converging cycle;
- If $r_{\mathsf{cycle}} \geq r_{\mathsf{prefix}}$,
  - then $\inf(\mathsf{Freq}(\mathcal{A})) = r_{\mathsf{cycle}}$ and it is reached;

– else $\inf(\mathsf{Freq}(\mathcal{A})) = r_{\mathsf{prefix}}$ and it is reached if and only if it is reached by a run whose contraction is a lasso around a simple cycle
(and one can decide whether it is the case using Lemma 6.1).

Let us now prove that this approach is correct.

First of all, if there is no reward-converging (simple) cycle in $\mathcal{A}_{cp}$, then all runs in $\mathcal{A}_{cp}$ are reward-diverging. For each run π, there exists a non-Zeno run ϱ in $\mathcal{A}$ with $\mathsf{Rat}(\pi) = \mathsf{freq}_{\mathcal{A}}(\varrho)$, thanks to Proposition 6.2. In this case, Lemma 6.4 allows us to conclude.

Assume now that $\mathcal{A}_{cp}$ contains a reward-converging cycle, and let S_{rc} be the set of states in $\mathcal{A}_{cp}$ that belong to a reward-converging cycle. The set S of cycle-free finite runs in $\mathcal{A}_{cp}$ ending in a state of S_{rc} is finite and therefore contains a run π_{prefix} with minimal ratio r_{prefix}. The infimum r^* of the ratios of runs of $\mathcal{A}_{cp}$ is thus $\min(r_{\mathsf{prefix}}, r_{\mathsf{cycle}})$ where r_{cycle} is the minimal ratio of reward-diverging simple cycles in $\mathcal{A}_{cp}$. Moreover, it is also the infimum of the frequencies of runs of $\mathcal{A}$ by Propositions 6.1, 6.2 and 6.3.

This bound is reached by a non-Zeno run of $\mathcal{A}$ whose contraction ends up staying in C if and only if $r_{\mathsf{cycle}} = r^*$.

Let us now assume that $r_{\mathsf{cycle}} > r^*$. The infimum may be reached only by a Zeno run. Let us prove that if a Zeno run ϱ reaches the bound, then there exists a run ϱ' reaching the bound and whose contraction is a reward-converging lasso. In this case, the contraction of ϱ is of ratio r_{prefix}. It is reward-converging, otherwise $r_{\mathsf{cycle}} > r^*$ would not hold. For the same reasons as in the proof of Lemma 6.1, it ends in non-accepting locations with a constant pointed region of the form $((k, k+1), -\bullet)$. Hence, there exists a run ϱ'' reaching $\inf(\mathsf{Freq}(\mathcal{A}))$ and whose contraction ends up iterating infinitely a simple cycle with states of the form $(\ell, (k, k+1), -\bullet)$ with $\ell \in \overline{F}$. Let us now consider the largest prefix this contraction such that the corresponding suffix is a lasso with only such states. This prefix has ratio r_{prefix} and it cannot iterate a reward-diverging cycle, because the ratio of the same prefix without the iteration of this cycle would be strictly smaller than r_{prefix} which is impossible. Let us consider the run π as the lasso obtained by removing all the iterations of cycles from the contraction that we consider. This run has ratio r_{prefix} because removed cycles have only rewards 0. Let us prove that it is a contraction and that there exists a run ϱ' whose contraction is π and such that $\mathsf{freq}_{\mathcal{A}}(\varrho') = r_{\mathsf{prefix}}$.

To do so, we are going to use the proof of Lemma 6.1. Let us recall the conditions for a run of $\mathcal{A}_{cp}$ to be a contraction:

- from each state of π of the form $(\ell, (i, i+1), \bullet\text{–})$ where $\ell \notin F$, π follows an idling transition to $(\ell, (i, i+1), -\bullet)$;
- after each move $(\ell, (i, i+1), \bullet\text{–}) \xrightarrow{1/1} (\ell, (i, i+1), -\bullet)$ where $\ell \in F$, π reaches $(\ell, \{i+1\}, \bullet)$ by an idling transition.

They cannot be satisfied thanks to additional reward-converging cycles. Hence, by construction of π from a contraction, π is a contraction. Then, recall the condition over the prefix of a contraction to have a run in $\mathcal{A}$ with a frequency equal to its ratio:

1. discrete transitions with resets go out of punctual regions or of the unbounded region;
2. discrete transitions going from F-locations to $\overline{F}$-locations or the opposite, has to be fired from a punctual region, or from the unbounded region after a positive reward.

Adding some cycles does not help to satisfy these conditions, therefore there exists a run ϱ' whose contraction is π and such that $\mathsf{freq}_{\mathcal{A}}(\varrho') = r_{\mathsf{prefix}}$.

Complexity In the corner-point abstraction, one can guess, in a non-deterministic way, either a minimal reward-diverging cycle, or a reward-converging lasso checking on-the-fly whether it is a contraction and whether its ratio is realizable. The size of the corner-point abstraction is linear in the size of the timed automaton when it has a single clock. Then, deciding whether there exists neither a reward-diverging cycle, nor a reward-converging lasso, whose ratio is smaller, is in co-NLOGSPACE. Now, since NLOGSPACE=co-NLOGSPACE, computing the bound $\inf(\mathsf{Freq}(\mathcal{A}))$ and deciding whether it is realized, can be done, in a non-deterministic way, in logarithmic space. Finally, the same reasoning holds for $\sup(\mathsf{Freq}(\mathcal{A}))$. □

Conclusion

We proved that the bounds of the set of the frequencies in a timed automaton with only one clock are the same as the bounds of the set of ratios in the corner-point abstraction, which can easily be computed. Furthermore, the set of frequencies of non-Zeno runs is equal to the set of ratios of reward-diverging runs, which is a finite union of intervals. Zeno runs can strongly alter the form of the set of frequencies, but the bounds are always the same as for the set of ratios in the corner-point abstraction, and one can decide if they are realized.

The link we established between the timed automaton and its corner-point abstraction differs in several aspects from what has been established in the general framework of double-priced timed automata in [BBL08]. First, a result similar to Proposition 6.1 was proven, but ours is more constructive. More precisely, the runs π and π' were not necessarily in $\mathsf{Proj}_{cp}(\varrho)$, and more importantly, it heavily relied on the reward-diverging hypothesis. Then the result which is comparable to Theorem 6.1 in [BBL08] is weaker, as there is no way to decide whether the bounds are realizable or not, and zenoness is not treated.

Along this chapter, we illustrated the limits of our techniques, by exhibiting counter-examples with two clocks. These example timed automata share a common aspect: they have convergence phenomena along a cycle. Cycles without such convergences are called forgetful and can be detected [BA11]. In the next chapter, we prove that, under the assumption that cycles are forgetful and that the time diverges, the set of frequencies in a timed automaton with several clocks is equal to the set of ratios in its corner-point abstraction.

Chapter 7

Frequencies in Forgetful Timed Automata

Introduction

A quantitative semantics for infinite timed words based on the notion of frequency has been introduced in Chapter 5. In Chapter 6, we presented how lower and upper bounds of the set of frequencies of one-clock timed automata can be computed using the corner-point abstraction, a refinement of the classical region abstraction, introduced in [BBL08]. Moreover, we showed that the realizability of these bounds is decidable, all this with NLOGSPACE complexity.

The techniques from Chapter 6 do not extend to timed automata with several clocks, and all counterexamples presented along the last chapter rely on some phenomenon of convergence between clocks along cycles. Beyond zenoness (when time converges along a run), other convergence phenomena between clocks were first discussed in [CHR02] in the context of control problems. Similarly to zenoness, these convergences correspond to behaviors that are unrealistic from an implementability point of view. A procedure to detect cycles with no such convergences (called forgetful cycles) has been recently introduced in [BA11]. This notion of forgetfulness was used to characterize timed languages with a non-degenerate entropy. An alternative notion of forgetfulness using the corner-point abstraction was also proposed in Chapter 5 and compared with the initial notion. Recall that in our definition, a cycle is said to be forgetful if its projection in the corner-point abstraction is strongly connected.

In this chapter, we naturally propose to investigate how forgetfulness can be exploited to compute frequencies. First, we show that forgetfulness of a cycle in a one-clock timed automaton is equivalent to not forcing the convergence of the clock, that is the clock is reset or not bounded. Note however that forgetfulness does not imply that all runs are non-Zeno. Under the assumption of forgetfulness, the set of frequencies for one-clock timed automata can be exactly computed, using the corner-point abstraction. Then, we show that in forgetful timed automata with several clocks, in which time diverges necessarily along a run, the set of frequencies can also be computed thanks to the corner-point abstraction. On the one hand, the result for timed automata for which all cycles are forgetful (strong forgetfulness) is as constructive as Proposition 6.2 in Chapter 6 over one-clock timed automata. On the other hand, to relax strong forgetfulness and consider forgetful and aperiodic timed automata, that is timed automata whose simple cycles and their powers are forgetful, the proof relies on a set of canonical runs whose set of frequencies agrees with the set of all frequencies in the timed automaton.

The chapter is structured as follows. In Section 7.1, we propose a characterizations of forget-

fulness in one-clock timed automata, and provide an expression for the set of frequencies for this restricted class. Then, Section 7.2 deals with timed automata with several clocks and explains how forgetfulness may be used to ensure that, when time diverges, the set of frequencies of a timed automaton and the set of ratios in its corner-point abstraction are equal.

7.1 Frequencies in one-clock forgetful timed automata

In this section, we improve on the results of Chapter 6 for the class of forgetful one-clock timed automata. Thanks to forgetfulness, we consider timed automata without cycles forcing Zeno behavior (*e.g.* a loop without reset and with the guard $x < 1$) which are not forgetful but we deal with cycles allowing Zeno behaviors which are forgetful. For this class of timed automata, we give an expression of the set of frequencies, whereas in Chapter 6 we dealt with all one-clock timed automata but only computing the bounds of the set of frequencies. Moreover, the class of forgetful one-clock timed automata is strictly larger than the class of strongly non-Zeno one-clock timed automata for which we gave an expression of the set of frequencies in Chapter 6.

One-clock timed automata have simpler clock behaviors than the general model. In fact, having a single clock in a timed automaton is quite close to forgetfulness in the sense that each time the clock is reset, the timed automaton forgets all the previous timing information. In this section, we present an equivalent characterization of forgetfulness in the case of one-clock timed automata, and we show the equivalence between forgetfulness and strong forgetfulness. Last, we propose an expression for the set of frequencies of forgetful one-clock timed automata.

In a one-clock timed automaton, a reset of the clock along a cycle is linked to forgetfulness. Indeed, the previous delays are forgotten at each reset of the clock, they do not impact on the current state and thus on the future states. The following lemma states the precise characterization of forgetful cycles inspired by this observation.

Proposition 7.1. *Let C be a cycle of a one-clock timed automaton. Then, C is forgetful if and only if the clock is reset or not bounded along C.*

Proof of Proposition 7.1. Let $\mathcal{A}$ be a timed automaton with a single clock x:

$\Rightarrow$: Let us prove the contrapositive of the left-to-right implication. Let C be a cycle in which x is bounded and not reset. Then, any region guard r along C is bounded and not punctual (only positive delays are allowed, see the explanations of the splitting in regions Section 5.1). Let α and $\alpha + 1$ be the two corners of r. There is no reset along C hence α is not reachable from $\alpha + 1$ in $\mathsf{Proj}(C)$. Thus, $\mathsf{Proj}(C)$ is not strongly connected and the cycle C is not forgetful.

$\Leftarrow$: Let us prove the right-to-left implication by inspecting two cases. Let C be a cycle and let us consider two cases:

- if x is not bounded, then, all the corners are $\perp$. The projection of C is thus trivially strongly connected, hence C is forgetful;
- if x is reset, then, let $s = (\ell, \{0\}, 0)$ be a state of $\mathsf{Proj}(C)$. From s, all the states of $\mathsf{Proj}(C)$ are reachable, because there is a single clock (regions have at most two corners, and the second corner can be reached from the first one by one idling transition). Moreover, s is reached at each iteration of the cycle C. Hence, the projection of C is strongly connected, thus C is forgetful. □

In fact, Proposition 7.1 implies that any cycle obtained by concatenation of forgetful cycles in a one-clock timed automaton is forgetful. Indeed, if the clock is reset or not bounded along each cycle of the concatenation, it is clearly the case for the concatenation itself.

Corollary 7.1. *A one-clock timed automaton is forgetful iff it is strongly forgetful.*

Recall that, as illustrated by the timed automaton in Figure 5.9 on page 130 which is not strongly forgetful, Corollary 7.1 does not hold for timed automata with several clocks.

Let us now consider the set of frequencies in a one-clock timed automaton. By Proposition 6.4, if there are only non-Zeno runs in a timed automaton, then the set of frequencies equals to the set of ratios in the corner-point. Firstly, the particular case where a timed automaton has a reachable reward-converging cycle in its corner-point containing both accepting and non-accepting locations (called mixed cycle) is easy to treat as stated in the following proposition.

Proposition 7.2. *Let $\mathcal{A}$ be a forgetful one-clock timed automaton. If there is a mixed reward-converging cycle in its corner-point $\mathcal{A}_{cp}$, then* $\mathsf{Freq}_Z(\mathcal{A}) =]0,1[$ *and* $\mathsf{Freq}_{nZ}(\mathcal{A}) = [0,1]$.

Proof of Proposition 7.2. The considered cycle belongs to the projection of a cycle C which is forgetful, then by Proposition 7.1, either x is reset or x is not bounded along C. As the guards of C are some regions and there is a reward-converging cycle in $\mathsf{Proj}(C)$, either x is reset and all the guards are $0 < x < 1$ (there is no guard $x = 0$ because zero delays are forbidden) or all the guards are $x > M$ (if x is not bounded). Then, the accumulated delay can be as small as necessary, but a delay close to 1 can be elapsed at each iteration in both cases. Moreover, this delay can be elapsed in any location along C. As a consequence, by alternating delays in accepting and non-accepting locations with the appropriate proportions before ending by a very small accumulated delay, we can construct a Zeno run with any frequency in $]0,1[$.

For the same reason, two runs respectively maximizing and minimizing the ratio along C (dilating and contracting) are reward-diverging and are respectively of frequency 1 and 0. Note that if all guards are $x > M$, to build such two runs, we can consider runs where the reward elapsed in each state is bounded by one, otherwise the dilatation (or contraction) leads to unbounded rewards in a single state and thus prevents the construction of the next discrete transition. Thanks to Proposition 6.4 which expresses the set of frequencies of non-Zeno runs, the set of frequencies $\mathsf{Freq}_{nZ}(\mathcal{A})$ is equal to $[0,1]$. To conclude, note that frequencies 0 and 1 are not possible for Zeno runs because zero delays are forbidden, cycles in $\mathsf{Proj}(C)$ are mixed and no delay can be neglected in Zeno runs. □

Now, for the general case, it is possible to consider only timed automata whose corner-point abstraction do not have such cycles in their corner-point. Then, all the reward-converging cycles in the corner-point abstractions have either only accepting states, or only non-accepting states. Such cycles are said to respectively be accepting and non-accepting. We can now give a general expression for the set of frequencies of a forgetful one-clock timed automaton.

For readability, let us define some notations. Given C a cycle of $\mathcal{A}$ having a reward-converging cycle in its projection, we write $p(C)$ for the set of ratios of cycle-free prefixes ending in reward-converging cycles of $\mathsf{Proj}(C)$ and $c(C)$ for the set of ratios of co-reachable reward-diverging cycles. By convention, we let $\max(\emptyset) = -1$ and $\min(\emptyset) = 2$. Then, we define $M(C) = \max(p(C) \cup c(C))$ and $m(C) = \min(p(C) \cup c(C))$,

$M(\mathcal{A}_{cp}) = \max\{M(C) \mid C$ accepting cycle of $\mathcal{A}$ with a reward-converging cycle in $\mathsf{Proj}(C)\}$ and

$m(\mathcal{A}_{cp}) = \min\{m(C) \mid C \text{ non-accepting cycle of } \mathcal{A} \text{ with a reward-converging cycle in } \mathsf{Proj}(C)\}$.

Moreover, a cycle is said *accepting* (resp. *non-accepting*) if it contains only accepting (resp. non-accepting) locations.

Theorem 7.1. *Let $\mathcal{A}$ be a forgetful one-clock timed automaton. If there is a mixed reward-converging cycle in $\mathcal{A}_{cp}$, then $\mathsf{Freq}_Z(\mathcal{A}) =]0,1[$ and $\mathsf{Freq}_{nZ}(\mathcal{A}) = [0,1]$. Otherwise: $\mathsf{Freq}(\mathcal{A}) = \mathsf{Rat}_{r-d}(\mathcal{A}_{cp}) \cup \big[0, M(\mathcal{A}_{cp})\big[\ \cup \ \big]m(\mathcal{A}_{cp}), 1\big]$.*

Proof. The first part of Theorem 7.1 is established in Proposition 7.2, let us now assume that there is no mixed reward-converging cycles in $\mathcal{A}_{cp}$. By Proposition 6.4, for non-Zeno runs: $\mathsf{Freq}_{nZ}(\mathcal{A}) = \mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$. The rest of the proof is based on the following lemma dealing with accepting and non-accepting reward-converging cycles.

Lemma 7.1. *Let C be a cycle of a one-clock forgetful timed automaton $\mathcal{A}$:*

- *If $\mathsf{Proj}(C)$ contains a non-accepting reward-converging cycle, then the set of frequencies of the infinite runs of $\mathcal{A}$ ending in C is $\big[0, M(C)\big[$.*
- *If $\mathsf{Proj}(C)$ contains an accepting reward-converging cycle , then the set of frequencies of the infinite runs of $\mathcal{A}$ ending in C is $\big]m(C), 1\big]$.*

Proof of Lemma 7.1. Let C be a cycle such that $\mathsf{Proj}(C)$ contains a non-accepting reward-converging cycle. C being forgetful and zero delays being forbidden, there must be a reward-diverging cycle in $\mathsf{Proj}(C)$. In particular, there are some non-Zeno runs ending in C and they all have frequency 0. In the sequel of the proof, we then consider only Zeno runs. Let ϱ be a Zeno run in $\mathcal{A}$ ending in C. The cycle C is forgetful, hence $\mathsf{Proj}(C)$ is strongly connected. Let ϱ_0 be a prefix of ϱ having a run in its projection which ends in a state (ℓ, r, α) of a reward-converging cycle in $\mathsf{Proj}(C)$. Note that, in this case, any state (ℓ, r, α') belongs to a reward-converging cycle of $\mathsf{Proj}(C)$ too. In particular, the dilatation of ϱ_0, which is the run in $\mathsf{Proj}(\varrho_0)$ which maximizes the ratio, ends in such a state. The dilatation thus has a ratio smaller than $M(C)$. As a consequence of Proposition 6.1, ϱ_0 has a frequency smaller than $M(C)$. The locations of C are non-accepting, hence the frequency of ϱ is strictly smaller than $M(C)$ too.

Moreover for any $\varepsilon > 0$, a prefix π_0^ε ending in a reward-converging cycle of $\mathsf{Proj}(C)$ and such that $M(C) - \varepsilon < \mathsf{Rat}(\pi_0^\varepsilon) \leq M(C)$ can be built by definition of $M(C)$. Then, π_0 can be mimicked up to ε by a prefix ϱ_0^ε (Proposition 6.3) and then this prefix can be prolongated with an accumulated delay smaller than ε to obtain an infinite run ϱ_ε ending in C and having a projection ending in a reward-converging cycle of $\mathsf{Proj}(C)$. Such a run can thus be constructed with a frequency as close as necessary to $M(C)$. As a consequence, the value $M(C)$ is the strict upper bound of the frequencies of infinite runs ending in C.

Finally, we can construct an infinite run with any frequency in $\big[0, M(C)\big[$ by iterating C as much as necessary with delays close to 1 (C is forgetful hence any delay $0 < d < 1$ is possible at each iteration) in order to decrease the frequency as much as necessary.

The second item of the Lemma 7.1 can be proved in the same way. □

Back to the proof of the second part of Theorem 7.1, the inclusion from right to left is straightforward from the non-Zeno case and Lemma 7.1.

Thanks to the equality in the non-Zeno case, the inclusion from left to right is only needed for the subset $\mathsf{Freq}_Z(\mathcal{A})$. Let thus ϱ be a Zeno run. It can be projected on a reward-converging run in the

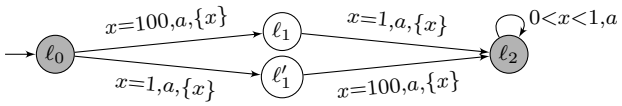


Figure 7.1: A non-forgetful counterexample to Theorem 7.1.

corner-point. This projection necessarily ends in a strongly connected subgraph G of the corner-point, because cycles are forgetful. Moreover, this subgraph has zero rewards and contains either (i) only accepting locations, or (ii) only non-accepting locations, because we assumed that there is no mixed cycle in the corner-point abstraction.

We study the case (i), the case (ii) is symmetric. As shown in the proof of Lemma 7.1, the prefix of ϱ corresponding to the prefix of the projection before the infinite suffix in the subgraph G has a frequency smaller than $M(C)$ for a cycle C having a reward-converging projection. To conclude, the frequency of ϱ is smaller than the one of the prefix because all the locations of the suffix are non-accepting. □

Remark 7.1. *The above theorem yields an expression of the set of frequencies which can be computed in polynomial time, whereas without the forgetfulness assumption, we only are able to compute the bounds of the set. Nevertheless, the complexity to compute the bounds of the set of frequencies is not improved with respect to Theorem 6.1, which states that, for one-clock timed automata, they can be computed by a non-deterministic procedure in logarithmic space.*

A non-forgetful counterexample to Theorem 7.1. Note that if the timed automaton is not forgetful, the form of the set of frequencies can be very different from the expression given in Theorem 7.1. Figure 7.1 gives an example of non-forgetful timed automaton such that $\mathsf{Freq}(\mathcal{A}) = \left]\frac{1}{101}, \frac{2}{102}\right[\cup \left]\frac{100}{101}, \frac{101}{102}\right[$. There is no reward-diverging run in $\mathcal{A}_{cp}$, $M(\mathcal{A}_{cp}) = -1$ because there is no accepting reward-converging cycle in $\mathcal{A}_{cp}$ and $m(\mathcal{A}_{cp}) = \frac{1}{101}$, hence the expected set of frequencies would be $]\frac{1}{101}, 1]$. The difference with forgetful timed automata is that the accumulated delays in ℓ_2 cannot diverge, therefore it is not possible to increase the frequency as much as necessary. In particular, there is no infinite run of frequency 1. More generally, this example illustrates a simple manner to obtain, for the set of frequencies, any finite union of open intervals included in $[0, 1]$. Nevertheless, Theorem 6.1 in Section 6.3 whose hypothesis and conclusion are weaker applies to such examples.

7.2 Extension to several clock forgetful timed automata

There is a real complexity gap between one-clock timed automata and timed automata with several clocks. For example, in [LMS04], the reachability problem for one-clock timed automata is proved to be in NLOGSPACE-complete, whereas it becomes PSPACE-complete with two clocks or more [AD94, FJ13]. As an other example, the language inclusion problem which is undecidable in the general case [AD94], becomes decidable with at most one clock [OW04].

In this section, we use forgetfulness and time divergence to compute the set of frequencies in timed automata with several clocks. Note that these assumptions are strong but can be justified by implementability concerns.

We aim to find some reasonable assumptions to obtain a class of timed automata whose sets of frequencies are exactly sets of ratios of their corner-point abstractions. More precisely, we want to extend the result $\mathsf{Freq}_{nZ}(\mathcal{A}) = \mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$ of Chapter 6, from one-clock timed automata to timed automata with several clocks. We do not want to complexify our problem dealing with Zeno runs as we did in one-clock timed automata for which the Zeno case is already non-trivial. As a consequence, we first assume that timed automata are strongly non-Zeno [AMPS98], that is, in every cycle there is one clock which is reset and lower guarded by a positive constant. This implies that there are no reward-converging runs in its corner-point. In [BBL08], double-priced timed automata are assumed to be strong reward-diverging which also implies that there are no reward-converging runs in the corner-point abstraction. For one-clock timed automata, strong non-zenoness is strictly stronger than forgetfulness and implies that $\mathsf{Freq}(\mathcal{A}) = \mathsf{Freq}_{nZ}(\mathcal{A}) = \mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$. Unfortunately, this assumption is not sufficient for timed automata with several clocks. For example, the timed automaton in Figure 6.7(a) in Section 6.2, is strongly non-Zeno and such that $\mathsf{Freq}(\mathcal{A}) =]0,1] \neq \{0\} \cup \{1\} = \mathsf{Rat}(\mathcal{A}_{cp})$. In fact, this timed automaton is a typical example of non-forgetful timed automaton. Delays in ℓ_1 have to be larger and larger along cycles, which ensures that frequency 0 cannot be reached in $\mathcal{A}$. On the contrary, in $\mathcal{A}_{cp}$, either the accumulated reward in ℓ_1 is always 0 (ratio 0), or there is one idling transition with reward 1 from a state of $\mathcal{A}_{cp}$ with location ℓ_1, and in the future, there always are such transitions in states of the form (ℓ_1, R, α) (ratio 1). Therefore, except over one-clock timed automata, forgetfulness and strong non-zenoness are not comparable.

7.2.1 Inclusion of the set of frequencies in the set of ratios

The main goal of this chapter is to prove the inclusion of the set of ratios in the corner-point in the set of frequencies in the timed automaton. This section is devoted to prove the other inclusion. This is based on a proposition from [BBL08] and will be a first illustration of the utility of forgetfulness to compute the set of frequencies in timed automata with several clocks.

Theorem 7.2. *Let $\mathcal{A}$ be a strongly non-Zeno and forgetful timed automaton. Then* $\mathsf{Freq}(\mathcal{A}) \subseteq \mathsf{Rat}(\mathcal{A}_{cp})$.

Proof. In this proof, we use the following proposition proved in [BBL08].

Proposition A ([BBL08])**.** *Let $\mathcal{A}$ be a strongly non-Zeno timed automaton, and let ϱ be an infinite run in $\mathcal{A}$. Then, the infinite run π consisting in the infinite iteration of the cycle of maximal ratio in $\mathcal{A}_{cp}$ is such that* $\mathsf{Rat}(\pi) \geq \mathsf{freq}_{\mathcal{A}}(\varrho)$.

Symmetrically, the infinite run consisting in an infinite iteration of the cycle of minimal ratio in $\mathcal{A}_{cp}$ has a ratio smaller than the frequency of any infinite runs in $\mathcal{A}$.

Let S_i ($i \in I$) be an SCC of $\mathcal{A}_{cp}$. Writing m_i (resp. M_i) the minimum (resp. maximum) of the ratios of cycles in S_i, we have $\mathsf{Rat}_{r-d}(\mathcal{A}_{cp}) = \bigcup_I [m_i, M_i]$ (Theorem 5.1). Also, since $\mathcal{A}$ is strongly non-Zeno $\mathsf{Rat}(\mathcal{A}_{cp}) = \mathsf{Rat}_{r-d}(\mathcal{A}_{cp})$. Hence using Proposition A, if $m = \min_I m_i$ and $M = \max_I M_i$, then $\mathsf{Freq}(\mathcal{A}) \subseteq [m, M]$.

Let us first consider the case where $\mathcal{A}_{cp}$ has a single strongly connected component (SCC for short). The set of ratios of $\mathcal{A}_{cp}$ is thus the interval $[m, M]$ where m and M are respectively the minimal and maximal ratios for a cycle of $\mathcal{A}_{cp}$. Therefore, the set of frequencies of $\mathcal{A}$ is a subset of the set of ratios in $\mathcal{A}_{cp}$.

The general case with several SCC is more complex because $\mathsf{Rat}(\mathcal{A}_{cp})$ is not convex *a priori*. The key is forgetfulness. Let ϱ be an infinite run of $\mathcal{A}$. Let π and π' be two infinite runs of $\mathcal{A}_{cp}$ in $\mathsf{Proj}(\varrho)$, and S and S' the respective SCC of $\mathcal{A}_{cp}$ in which they end. The run ϱ being infinite, there is a simple

cycle of $\mathcal{A}$ which is visited infinitely often. This cycle is forgetful by assumption, hence its projection is strongly connected and $S = S'$. Now, let us consider $\mathcal{B}$ the sub-automaton of $\mathcal{A}$ containing only the SCC in which ϱ ends. The prefix of ϱ can be neglected in the computation of the frequency of ϱ, since $\mathcal{A}$ is strongly non-Zeno. Hence $\mathsf{freq}_{\mathcal{A}}(\varrho) \in \mathsf{Rat}(\mathcal{B}_{cp}) \subseteq \mathsf{Rat}(\mathcal{A}_{cp})$. □

In the sequel, we see how strongly non-zenoness and forgetfulness can be useful to obtain the inclusion $\mathsf{Rat}(\mathcal{A}_{cp}) \subseteq \mathsf{Freq}(\mathcal{A})$. The problem is not trivial even under these assumptions and the proof techniques could certainly be interesting in other contexts. This section allows to understand several subtleties of forgetfulness.

7.2.2 Techniques to compute the frequencies

In this section, we explain the technical aspects which allow to deal with timed automata with several clocks. To begin with basis, there are two important lemmas respectively due to [BBL08] and [Pur00].

The following lemma, due to [BBL08], establishes that if a valuation v is ε-close to a corner α, any transition $(\ell, R, \alpha) \to (\ell', R', \alpha')$ of the corner-point abstraction can be mimicked in $\mathcal{A}$ by a transition $(\ell, v) \to (\ell', v')$ in the timed automaton, with v' ε-close to the corner α'.

Let us define some notations to formalize the distance to a corner.

$$\mu_v(x) = \min\{|v(x) - p| \mid p \in \mathbb{N}\},$$

$$\nu_v(x, y) = \min\{|v(x) - v(y) - p| \mid p \in \mathbb{N}\},$$

$$\delta(v) = \max_{x,y \in X}(\max(\{\mu_v(x)\} \cup \{\nu_v(x, y)\})).$$

The function δ, thus associate with each valuation, the distance to the closest corner.

Lemma 7.2 ([BBL08]). *Consider a transition $(\ell, R, \alpha) \to (\ell', R', \alpha')$ in $\mathcal{A}_{cp}$, take a valuation $v \in R$ such that $\delta(v) < \varepsilon$ and $|v(x) - \alpha(x)| = \mu_v(x)$. There exists a valuation $v' \in R'$ such that $(\ell, v) \to (\ell', v')$ in $\mathcal{A}$, $\delta(v') < \varepsilon$ and $|v'(x) - \alpha'(x)| = \mu_{v'}(x)$.*

In [BBL08], clocks are bounded, hence unbounded regions are not considered. Nevertheless, the same result holds for unbounded regions, considering abstract valuations instead of the corner in the same way as in the proof of Lemma C. Lemma 7.2 thus implies by induction on the length of the run, that any run in $\mathcal{A}_{cp}$ can be mimicked in $\mathcal{A}$ up to any $\varepsilon > 0$. As a consequence, respective lower and upper bounds of the sets of ratios and frequencies are equal, but as seen with the timed automaton in Figure 6.7(a), $\mathsf{Freq}(\mathcal{A})$ can be very different from $\mathsf{Rat}(\mathcal{A}_{cp})$ when $\mathcal{A}$ is not forgetful.

On the other hand, Lemma 7.3 expresses the preservation of barycentric relations between valuations along transitions.

Lemma 7.3 ([Pur00]). *Let (ℓ, g, a, X', ℓ') be an edge of $\mathcal{A}$ and v, v', w and w' be some valuations of X such that $(\ell, v) \to (\ell', v')$ and $(\ell, w) \to (\ell', w')$ with $R(v) = R(w)$ and $R(v') = R(w')$, then for any $\lambda \in [0, 1]$ $(\ell, \lambda v + (1 - \lambda)w) \to (\ell', \lambda v' + (1 - \lambda)w')$.*

Naturally, this lemma can be extended to finite sequences of edges by induction and, in particular, to cycles.

The combination of both lemmas helps us to prove that if along a given cycle one can go from every corner to a fixed corner α, then along this cycle one can go as close to α as necessary in $\mathcal{A}$. This way of reducing the distance to corners is the key to deal with timed automata with several clocks.

Indeed, given a run π of the corner-point abstraction, it allows to build a run ϱ mimicking it more and more precisely, *i.e.* for all ε there is a suffix of ϱ which mimics the corresponding suffix of π up to ϵ. Moreover, when time diverges (non-zenoness), if an infinite run ϱ in $\mathcal{A}$ mimics an infinite run π of $\mathcal{A}_{cp}$ up to ε converging to 0 along ϱ, then $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$.

Lemma 7.4. *Let $\mathcal{A}$ be a timed automaton and $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0,a_0} (\ell_1, v_1) \xrightarrow{\tau_1,a_1} \cdots \xrightarrow{\tau_{n-1},a_{n-1}} (\ell_0, v_n)$ with $r := R(v_0) = R(v_n)$ a bounded region, be a finite run of $\mathcal{A}$. Given a corner α_n of the region r, if for any (ℓ_0, r, α) there is a finite run from (ℓ_0, r, α) to (ℓ_0, r, α_n) in $\mathsf{Proj}(\varrho)$, then for all $\varepsilon > 0$, there exists $\varrho' = (\ell_0, v_0) \xrightarrow{\tau'_0,a_0} (\ell_1, v'_1) \cdots \xrightarrow{\tau'_{n-1},a_{n-1}} (\ell_0, v'_n)$ such that $\mathsf{Proj}(\varrho') = \mathsf{Proj}(\varrho)$ and $||v'_n - \alpha_n|| < \varepsilon$.*

Proof of Lemma 7.4. Let us start by fixing, for all corners α, a valuation v^ε_α in r which is ε-close to α. Thanks to these valuations, we then define a barycentric expression for v_0. Let Ω_r be the set of corners of r. As the closure of r is the convex hull of Ω_r (for the usual topology of $\mathbb{R}^{|X|}$), there exists $(v^\varepsilon_\alpha \in r)_{\alpha\in\Omega_r}$ and $(\lambda_\alpha \in [0,1])_{\alpha\in\Omega_r}$ such that $\sum_{\alpha\in\Omega_r} \lambda_\alpha = 1$, $v_0 = \sum_{\alpha\in\Omega_r} \lambda_\alpha v^\varepsilon_\alpha$ and $||v^\varepsilon_\alpha - \alpha|| < \varepsilon$. By assumptions, there are some paths in $\mathsf{Proj}(\varrho)$ going from each α to α_n. Thanks to Lemma 7.2, there are some finite runs from each of our valuations v^ε_α very close to the corner α to some valuation $v^\varepsilon_{\alpha,\alpha_n}$ very close to a common corner α_n. Formally, there exists $(v^\varepsilon_{\alpha,\alpha_n} \in r)_{\alpha\in\Omega_r}$ and some finite runs $(\varrho_\alpha)_{\alpha\in\Omega_r}$ from $(\ell_0, v^\varepsilon_\alpha)$ to $(\ell_0, v^\varepsilon_{\alpha,\alpha_n})$ in $\mathcal{A}$ with $||v^\varepsilon_{\alpha,\alpha_n} - \alpha_n|| < \varepsilon$. Then, by Lemma 7.3, there is a finite run ϱ' from (ℓ_0, v_0) to the state with location ℓ_0 and the valuation equal to the barycenter of the valuations $v^\varepsilon_{\alpha,\alpha_n}$ which is very close to α_n by the triangle inequality. Formally, there is a finite run ϱ' from $(\ell_0, v_0 = \sum_{\alpha\in\Omega_r} \lambda_\alpha v^\varepsilon_\alpha)$ to $(\ell_0, \sum_{\alpha\in\Omega_r} \lambda_\alpha v^\varepsilon_{\alpha,\alpha_n})$. To conclude, ϱ' is as needed because, by the triangle inequality, $||\sum_{\alpha\in\Omega_r} \lambda_\alpha v^\varepsilon_{\alpha,\alpha_n} - \alpha_n|| \leq \sum_{\alpha\in\Omega_r} \lambda_\alpha ||v^\varepsilon_{\alpha,\alpha_n} - \alpha_n|| < \varepsilon$. □

Remark that we assumed that region r is bounded. Lemma 7.4 does not hold, replacing corners by abstract valuations, if r is unbounded. Indeed, if along the path, there is a clock x which is reset and guarded by $x = 1$ at each transition and a clock y which is unbounded, then the distance between y and the abstract valuation cannot decreases. We are going to see in the sequel, that it does not prevent our reasoning to hold because intuitively, only delays impact on frequencies. We can thus reason without considering clock y, in this case, because it does not constraint delays.

To use Lemma 7.4, we need to find a cycle in $\mathcal{A}_{cp}$ which allows, intuitively, to synchronize all the corners of a region to a common one. Indeed, each run in $\mathcal{A}_{cp}$ corresponds to a run in $\mathcal{A}$, the existence of ϱ is not a real constraint. Moreover, Lemma 7.4 does not require forgetfulness of timed automata. The following lemma illustrates how forgetfulness can help to use Lemma 7.4. More precisely, it establishes that given a long enough sequence of cycles such that the concatenation of any subsequence of cycle is forgetful, then one can go from any corner to any corner following the sequence of cycle.

Lemma 7.5. *Let $\mathcal{A}$ be a timed automaton, X its set of clocks and a sequence $(c_i)_{1\leq i\leq K}$ with $K = 2^{|X|+1}$, of forgetful cycles containing the location ℓ of $\mathcal{A}$ such that all the cycles obtained by concatenation of the cycles of a subsequence $(c_k)_{i\leq k\leq j}$ with $1 \leq i$ and $j \leq K$, are forgetful. Then for all pairs of corners (α, α') of the region R associated to ℓ, there is a finite run of the form $(\ell, R, \alpha) \xrightarrow{\pi_1} (\ell, R, \alpha_1) \xrightarrow{\pi_2} \cdots (\ell, R, \alpha_{K-1}) \xrightarrow{\pi_K} (\ell, R, \alpha')$ such that for all indices i, π_i corresponds to one iteration of c_i.*

Proof of Lemma 7.5. Abusing notations we write $\pi \in \mathsf{Proj}(c)$ for "π corresponds to one iteration of c". Consider the subset construction with $s_0 = \{(\ell, R, \alpha)\}$ and $s_{i+1} = \{(\ell, R, \beta') \,|\, \exists (\ell, R, \beta) \in s_i, \exists \pi'_i \in \mathsf{Proj}(c_i), \text{ s.t. } (\ell, R, \beta) \xrightarrow{\pi'_i} (\ell, R, \beta')\}$.

First, there are at most $|X|+1$ corners in R, hence there are at most $K = 2^{|X|+1}$ subsets of $(\ell, R, \text{all}) := \{(\ell, R, \alpha) \mid \alpha \text{ corner of } R\}$. Second, by forgetfulness of the c_i's, if $s_i = (\ell, R, \text{all})$ then for all $j > i$, $s_j = (\ell, R, \text{all})$. Third, there is no other cycles in the subset construction. Indeed, if there exist indices $i < j$ such that $s_i = s_j \neq (\ell, R, \text{all}) := \{(\ell, R, \alpha) \mid \alpha \text{ corner of } R\}$ then the cycle obtained by concatenation of cycles $c_{i+1}, \cdots, c_j$ is not forgetful, which contradicts strong forgetfulness.

As a consequence, the subset construction loops in (ℓ, R, all) forever after a cycle-free prefix whose length is smaller than K. Hence, there is a finite run of the form $(\ell, R, \alpha) \xrightarrow{\pi_1} (\ell, R, \alpha_1) \xrightarrow{\pi_2} \cdots (\ell, R, \alpha_{K-1}) \xrightarrow{\pi_K} (\ell, R, \alpha')$ such that for all indices i, $\pi \in \mathsf{Proj}(c_i)$. □

In the next sections we use these two last lemmas to prove that strong non-zenoness and strong forgetfulness are sufficient to ensure the existence of such synchronizing cycles along infinite runs in the corner-point abstraction that we want to mimic them in $\mathcal{A}$.

7.2.3 Inclusion of the set of ratios in the set of frequencies

We first consider the case of strongly forgetful timed automata. Thanks to Lemma 7.4 and by observing the consequences of forgetfulness of all the cycles in a timed automaton, we obtain a theorem which is as constructive as Proposition 6.2 in Chapter 6 for one-clock timed automata.

Theorem 7.3. *Let $\mathcal{A}$ be a strongly non-Zeno strongly forgetful timed automaton. Then, for every infinite run π in the corner-point of $\mathcal{A}$, there exists an infinite run ϱ_π in $\mathcal{A}$ such that $\pi \in \mathsf{Proj}(\varrho_\pi)$ and $\mathsf{freq}_{\mathcal{A}}(\varrho_\pi) = \mathsf{Rat}(\pi)$.*

The idea is to prove, for every run π in $\mathcal{A}_{cp}$, the existence of synchronizing cycles infinitely often along π which allow to mimic it up to an ε converging to 0.

Proof. Along the infinite run π of $\mathcal{A}_{cp}$, there is a pair (ℓ, R) which appears infinitely often, possibly with different corners. Let $(\ell, R, \alpha_i)_{i\in\mathbb{N}}$ be a sequence of the occurrences of (ℓ, R) and $(\pi_i)_{i\in\mathbb{N}}$ the sequence of factors of π leading respectively from (ℓ, R, α_i) to (ℓ, R, α_{i+1}). Each π_i corresponds to a forgetful cycle c_i in $\mathcal{A}$ hence by Lemma 7.5, for all pairs (α, α') of corners of the region R, there is a finite run of the form $(\ell, R, \alpha) \xrightarrow{\pi'_1} (\ell, R, \alpha_1) \xrightarrow{\pi'_2} \cdots (\ell, R, \alpha_{K-1}) \xrightarrow{\pi'_K} (\ell, R, \alpha')$ with $K = 2^{|X|+1}$ and such that for all indices i, π_i corresponds to one iteration of c_i. In particular, this finite run belongs to the projections of exactly the same runs as $\pi_1 \cdot \pi_2 \cdots \pi_K$. As a consequence, for any finite run $\varrho = (\ell, v_0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \xrightarrow{\tau_1, a_1} \cdots \xrightarrow{\tau_{n-1}, a_{n-1}} (\ell, v_n)$ with $R(v_0) = R(v_n) = R$ and such that $\pi_0.\pi_1 \cdots \pi_K \in \mathsf{Proj}(\varrho)$, for any corner β_n of the region R and for all (ℓ, R, α) there is a finite run from (ℓ, R, α) to (ℓ, R, β_n) in $\mathsf{Proj}(\varrho)$.

Let first assume that region R is bounded. Hence, Lemma 7.4 can be applied to such finite runs. Then, for any ε and given ϱ^i a mimicking of π until (ℓ, R, α_i), Lemma 7.4 ensures the existence of a extension of ϱ^i to ϱ^{i+K} mimicking π until (ℓ, R, α_{i+K}), such that $||v - \alpha_{i+K}|| < \varepsilon$ where v is the last valuation of ϱ^{i+K}. In words, it is possible to define some finite factors along π which allow to go as close as necessary from a corner of π while mimicking ϱ. Out of these factors, the distance to the corners of π can be preserved thanks to Lemma 7.2. To conclude, these factors can be picked infinitely often to allow the convergence of the distance to the corner of π to 0, but as rarely as necessary to be neglected in the computation of the frequency.

Now, if a clock y is not bounded in R and it is reset in an infinite number of cycle c_i's. Then, the construction of ϱ can be done ignoring y when it is unbounded. As discussed after Lemma 7.4,

the distance of y to its abstract value can be not decreased arriving in R. Nevertheless, when y will be reset, then its distance to its abstract value is 0. As a consequence, the distance between the valuation and the corner is defined only by the other clocks and y can come back in the process of the construction. Thus y does not prevent the distance to the corner to decrease in the long term.

Finally, if a clock is only finitely reset, then one can consider a sequence of cycle after its last reset, and this clock can be ignored forever. Indeed, only delays impact on the computation of the frequency and this clock does not constraint them. □

Theorem 7.3 implies that the set of ratios $\mathsf{Rat}(\mathcal{A}_{cp})$ is included in the set of frequencies $\mathsf{Freq}(\mathcal{A})$. This implies, together with Theorem 7.2, that if $\mathcal{A}$ is a strongly non-Zeno and strongly forgetful timed automaton, then $\mathsf{Freq}(\mathcal{A})$ is equal to $\mathsf{Rat}(\mathcal{A}_{cp})$. Then, Corollary 5.1 states that its bounds can be computed in polynomial space.

Corollary 7.2. *Let $\mathcal{A}$ be a strongly non-Zeno strongly forgetful timed automaton. Then,* $\mathsf{Freq}(\mathcal{A}) = \mathsf{Rat}(\mathcal{A}_{cp})$ *and its bounds can be computed in polynomial space.*

Strong forgetfulness is a realistic assumption from an implementability point of view, but is not satisfactory because of its difficulties to be checked. Indeed, checking if a cycle is forgetful can be done in the corner-point abstraction, but there is an unbounded number of cycles in a timed automaton and we do not know how to avoid to check them all. As a consequence, it is important to relax this assumption. We did not succeed in proving that strong forgetfulness can be removed from the hypotheses of Theorem 7.3. Nevertheless, the inclusion $\mathsf{Rat}(\mathcal{A}_{cp}) \subseteq \mathsf{Freq}(\mathcal{A})$ still holds when strong forgetfulness is replaced by forgetfulness and aperiodicity (forgetfulness of the powers of simple cycles), both of which can be checked on the corner-point abstraction.

Theorem 7.4. *Let $\mathcal{A}$ be a strongly non-Zeno, forgetful and aperiodic timed automaton. Then, we have the following inclusion:* $\mathsf{Rat}(\mathcal{A}_{cp}) \subseteq \mathsf{Freq}(\mathcal{A})$.

Proof. We want to prove that for all $\mathsf{rat} \in \mathsf{Rat}(\mathcal{A}_{cp})$, there exists an infinite run π_{rat} in $\mathcal{A}_{cp}$ of ratio rat and such that there exists an infinite run ϱ_π of $\mathcal{A}$ with $\mathsf{freq}_{\mathcal{A}}(\varrho_\pi) = \mathsf{Rat}(\pi_{\mathsf{rat}})$ and $\pi_{\mathsf{rat}} \in \mathsf{Proj}(\varrho_\pi)$.

By Theorem 5.1, one knows an expression for the set of ratios of the corner-point abstraction: $\mathsf{Rat}(\mathcal{A}_{cp}) = \bigcup_{S_i \in SCC}[m_i, M_i]$, where m_i (resp. M_i) is the minimum (resp. maximum) of the ratios of cycles of the SCC S_i of the corner-point.

Let i be the index of an SCC, and a rational number $\mathsf{rat} \in [m_i, M_i]$. Then, one can build an infinite run π_{rat} in $\mathcal{A}_{cp}$ with ratio rat and ending in S_i by alternating iterations of a cycle c_i of ratio m_i and a cycle C_i of ratio M_i in S_i with the suitable proportion. The prefix to go to c_i and the finite run to go from a cycle to the other are neglected in the computation of the ratio by multiplying the number of iterations in both cycles at each step, by a common integer. If rat is an irrational number, there exists an increasing sequence of rationals converging to it, and the same construction can be done by following the successive proportions corresponding to the elements of the sequence, in the same way as in the proof of Theorem 5.1.

π_{rat} can be mimicked up to ε for any $\varepsilon > 0$ (Lemma 7.2). However this is not sufficient to ensure that there exists a run ϱ_π of $\mathcal{A}$ with $\mathsf{freq}_{\mathcal{A}}(\varrho_\pi) = \mathsf{Rat}(\pi_{\mathsf{rat}})$ and $\pi_{\mathsf{rat}} \in \mathsf{Proj}(\varrho_\pi)$. Then, we extend in π_{rat}, the finite run π_{C_i,c_i} to go from C_i to c_i by a finite run $\pi_{c_i^K}$ with $K = 2^{|X|+1}$ which iterates K times c_i in $\mathcal{A}_{cp}$. We thus want to prove that it allows to mimic π_{rat} up to an ε converging to 0. This finite extension has a constant length and will be neglected in the computation of the ratio in the same way as π_{C_i,c_i} even if it means to increase the numbers of iterations of C_i and c_i at each step.

The cycle c_i is in the projection of a cycle (simple or a concatenation of a single simple cycle) $\hat{c}_i$. $\hat{c}_i$ is forgetful and aperiodic by assumption (the concatenation of an aperiodic cycle is aperiodic), that is all the concatenations of $\hat{c}_i$'s are forgetful. Then, thanks to Lemma 7.5, for all pairs of states (ℓ, R, α) and (ℓ, R, α') of $\mathsf{Proj}(\hat{c}_i)$, there is a run $\pi_{\alpha,\alpha'}$ from (ℓ, R, α) to (ℓ, R, α') corresponding to $2^{|X|+1}$ iterations of $\hat{c}_i$. In particular, $\pi_{\alpha,\alpha'}$ and $\pi_{c_i^{2^{|X|+1}}}$ belong to the projections of exactly the same runs. As a consequence, for any finite run $\varrho = (\ell, v_0) \xrightarrow{\tau_0,a_0} (\ell_1, v_1) \xrightarrow{\tau_1,a_1} \cdots \xrightarrow{\tau_{n-1},a_{n-1}} (\ell, v_n)$ iterating $2^{|X|+1}$ times $\hat{c}_i$, for any corner α_n of the region R associated to ℓ and for all (ℓ, R, α) there is a finite run from (ℓ, R, α) to (ℓ, R, α_n) in $\mathsf{Proj}(\varrho)$. In other words, Lemma 7.4 can be applied to such finite runs if R is bounded. Hence, building π_{rat} in $\mathcal{A}_{cp}$ as explain above and replacing the finite runs π_{C_i,c_i} to go from C_i to c_i by the concatenation $\pi_{C_i,c_i}\pi_{c_i^{2^{|X|+1}}}$, π_{rat} can be mimicked up to a decreasing ε each time that we go from C_i to c_i.

Finally, if R in unbounded, unbounded clocks can be ignored in the same way as explained at the end of the proof of Theorem 7.3 holds. □

We thus obtain the following result as a corollary of Theorems 7.2 and 7.4 and Corollary 5.1.

Corollary 7.3. *Let $\mathcal{A}$ be a strongly non-Zeno, forgetful and aperiodic timed automaton. Then, $\mathsf{Freq}(\mathcal{A}) = \mathsf{Rat}(\mathcal{A}_{cp})$ and its bounds can be computed in polynomial space.*

Strong forgetfulness is stronger than aperiodicity, hence Theorem 7.3 cannot help to establish this equality in a more general case. However, note that Theorem 7.4 does not imply Theorem 7.3. In Theorem 7.3, not only the inclusion $\mathsf{Rat}(\mathcal{A}_{cp}) \subseteq \mathsf{Freq}(\mathcal{A})$ is established, but also for all infinite runs π in $\mathcal{A}_{cp}$ there exists an infinite run ϱ in $\mathcal{A}$ with $\pi \in \mathsf{Proj}(\varrho)$ and $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$. In Theorem 7.4, this is only proved for some infinite runs π of $\mathcal{A}_{cp}$.

7.2.4 Discussion about assumptions

As explained above, our will to relax the strong forgetfulness is due to the difficulties to check this property. Strong forgetfulness clearly implies at once forgetfulness and aperiodicity, but a first open question is whether the opposite implication is true. We conjecture that if there is an example of forgetful aperiodic timed automata which is not strongly forgetful, then it has more than three clocks which make the search more complex.

An other open question is whether the hypothesis of aperiodicity in Theorem 7.4 can be removed. We use this hypothesis in the proof, but could not find examples of forgetful periodic timed automata for which Theorem 7.4 does not hold. We built some examples of periodic timed automata as in Figure 5.9, but periodic timed automata seem to be degenerated and in particular, based on punctual guards which implies bijections between runs in the timed automaton and those in its corner-point abstraction.

Conclusion

In this chapter, we used a notion of forgetfulness to extend the results about frequencies in timed automata. On the one hand, thanks to forgetfulness, we can compute the set of frequencies in one-clock timed automata, even with Zeno behaviors, whereas only the bounds of this set were computed in Chapter 6. On the other hand, with forgetfulness and time-divergence as in [BBL08], we compute

the set of frequencies in a class of timed automata with several clocks, whereas techniques of Chapter 6 were not applicable.

Our contribution can also be compared with that of [BBL08] presented in Chapter 5 on double priced timed automata, that is, timed automata with costs and rewards. Indeed, frequencies are a particular case of cost and reward functions. In [BBL08], either a run of minimal ratio or an optimal family (*i.e.* ε-optimal runs for all $\varepsilon > 0$) is computed, whereas here, assuming forgetfulness, the exact set of frequencies can be computed, not only the optimal ones. Our techniques might thus prove useful for double priced timed automata and maybe more generally in other contexts.

In future work, we would like to investigate more deeply the difference between forgetfulness and strong forgetfulness with the hope to extend Theorem 7.3. Moreover, Theorem 7.2 is less constructive than the equivalent result for one-clock timed automata which uses notions of contraction and dilatation of a run. It would be interesting to see if forgetfulness could help to extend these constructions to timed automata with several clocks. Finally, our main tool presented in Lemma 7.4 can easily be used for the scheduling problem in timed automata with costs and rewards studied in [BBL08]. Thus, we can prove that in strongly non-Zeno forgetful timed automata, there always exists an infinite run whose ratio is optimal. We hope that Lemma 7.4, which is fundamental, will be useful for other problems for which the corner-point abstraction is suitable.

Chapter 8

Emptiness and Universality Problems in Timed Automata with Frequency

Introduction

Earlier in this part, we introduced the notion of frequency as the proportion of time elapsed in accepting locations along a run. This allows us to define languages of timed automata, for example with positive frequency or using thresholds. In Chapters 6 and 7, we presented techniques to compute bounds of the set of frequencies in timed automata of two distinct classes. Our main motivation was to study languages and more specifically two main questions: "is the language empty ?" and "is the language universal?".

In this chapter, we draw the consequences of these results. The emptiness problem for timed automata satisfying one of the restrictions, can easily be decided using the bounds of the set of frequencies. Moreover, in the restricted case of deterministic timed automata, it allows to decide the universality problem for these languages. The universality problem for non-deterministic timed automata is much harder. We prove that it is non-primitive recursive for one-clock timed automata, and becomes undecidable with several clocks, even for the positive frequency semantics. The decidability status for one-clock timed automata with positive frequency is still open, but we show that the universality problem for Zeno words in one-clock timed automata with positive frequency acceptance is decidable. These latter problems remain open for the semantics with threshold.

This chapter is structured as follows. In Section 8.1, we present the consequences of the results of Chapters 6 and 7. Section 8.2 is devoted to the hardness of the universality problem. Last, we establish the decidability of the universality problem for Zeno words in one-clock timed automata with positive frequency in Section 8.3.

8.1 Consequences of Chapters 6 and 7

In Chapters 6 and 7 we proposed approaches to compute bounds of the set of frequencies and decide their realizability for timed automata of two classes: one-clock timed automata and strongly non-Zeno, forgetful and aperiodic timed automata. These results allows to decide language problems such as the emptiness problem.

The emptiness problem. In our context, the emptiness problem asks, given a timed automaton $\mathcal{A}$ whether there is an infinite timed word which is accepted by $\mathcal{A}$ under a given frequency-based constraint. As a consequence of Theorem 6.1, we get the following results.

Theorem 8.1. *The emptiness problem for infinite timed words in one-clock timed automata $\mathcal{A}$ is in NLOGSPACE.*

Thanks to Corollary 7.3, we can extend this result to timed automata with several clocks with a PSPACE complexity.

Theorem 8.2. *The emptiness problem for infinite timed words in strongly non-Zeno, forgetful and aperiodic timed automata $\mathcal{A}$ is in PSPACE.*

Upper threshold languages We defined lower threshold languages by accepting words for which there exists a run of ratio larger than a threshold λ. We can also define languages with an upper threshold, using as acceptance condition that the frequency of a word is smaller than a threshold λ. Since the frequency of a word is the minimum of the frequency of runs reading it, deciding the emptiness of such a language is harder than for lower threshold languages. Indeed, it can be deduced from the set of frequencies, only if the timed automaton is deterministic. However, the universality can be decided by comparing λ with the upper bound of the set of frequencies.

The universality problem. We now focus on the universality problem, which asks, whether all timed words are accepted under a given frequency-based acceptance condition in a given timed automaton. We also consider variants thereof which distinguish between Zeno and non-Zeno timed words. Note that, as presented in Section 5, these variants are incomparable: there are timed automata that, with positive frequency, recognize all Zeno timed words but not all non-Zeno timed words, and *vice-versa*.

A first obvious result concerns deterministic timed automata. One can first check syntactically whether all infinite timed words can be read (just locally check that the automaton is complete). Then we notice that considering all timed words exactly amounts to considering all runs. Thanks to Theorem 6.1 for one-clock timed automata, one can decide, in this case, whether there is or not a run of frequency does not satisfy the frequency-based constraint. The existence of such a run is equivalent to the non-universality of the timed automaton.

Theorem 8.3. *The universality problem for infinite (resp. non-Zeno, Zeno) timed words in deterministic one-clock timed automata is in NLOGSPACE.*

In the same way, Corollary 7.3 allows to decide the universality problem in deterministic, strongly non-Zeno, forgetful and aperiodic timed automata.

Theorem 8.4. *The universality problem for infinite (resp. non-Zeno, Zeno) timed words in deterministic, strongly non-Zeno, forgetful and aperiodic timed automata is in PSPACE.*

Remark that both above theorems also hold for upper threshold languages.

8.2 Lower bound for the universality problem

In the previous section we saw that results of Chapters 6 and 7 allow one to decide universality for some deterministic timed automata. In this section, we prove that if we relax the assumption of determinism this becomes much harder!

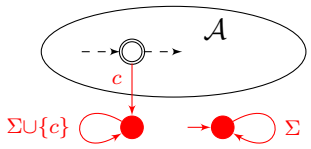


Figure 8.1:

Figure 8.2: Reduction for the proof of Theorem 8.5.

Theorem 8.5. *The universality problem for infinite (resp. non-Zeno, Zeno) timed words in a one-clock timed automaton is non-primitive recursive. If two clocks are allowed, this problem is undecidable.*

Proof. The proof is done by reduction from the universality problem for finite words in timed automata (which is known to be undecidable for timed automata with two clocks or more [AD94] and non-primitive recursive for one-clock timed automata [OW04]). Given a timed automaton $\mathcal{A}$ that accepts finite timed words, we construct a timed automaton $\mathcal{B}$ with an extra letter c which will be interpreted with positive frequency. From all accepting locations of $\mathcal{A}$, we allow $\mathcal{B}$ to read c and then accept everything (with positive frequency). The construction is illustrated on Fig. 8.2. It is easy to check that $\mathcal{A}$ is universal for finite timed words over Σ iff $\mathcal{B}$ is universal for infinite (resp. non-Zeno, Zeno) timed words over $\Sigma \cup \{c\}$. □

Remark that we do not know wether the universality problem for forgetful timed automata with two clocks is still undecidable

8.3 Decidability of the universality problem for Zeno words with positive frequency in one-clock timed automata

The universality problem for one-clock timed automata with positive frequency is proved to be non-primitive recursive in Section 8.2. Its decidability stays open, but distinguishing the Zeno and non-Zeno cases we obtain a partial answer. Whereas in general, the non-zenoness assumption simplifies most problems, here the Zeno case is decidable and the decidability of the non-Zeno case is still open.

Theorem 8.6. *The universality problem for Zeno timed words with positive frequency in a one-clock timed automaton is decidable.*

The proof of this result is nice and mainly due to Patricia Bouyer-Decître. As a consequence, we only give a sketch here. The complete proof can be find in [BBBS13].

Sketch of proof. Given a timed automaton $\mathcal{A}$, with a single clock, we want to check whether $\mathcal{A}$ is universal for Zeno timed words with positive frequency. We first check that every Zeno timed word can be read in $\mathcal{A}$: this is equivalent to checking that all finite timed words can be read in $\mathcal{A}$, and this can be done [OW04]. Thus, without loss of generality, we assume that $\mathcal{A}$ reads all Zeno timed words, and we now only need to take care of the accepting condition.

The proof is based on the idea that for a Zeno timed word to be accepted with positive frequency it is (necessary and) sufficient to visit an accepting location once. Furthermore the sequence of timestamps associated with a Zeno timed word is converging, and we can prove that from some point on,

in the automaton, all guards will be either verified or falsified: for instance if the value of the clock is 1.4 after having read a prefix of the word, and if the word then converges in no more than 0.3 time units, then only the constraint $1 < x < 2$ will be satisfied while reading the suffix of the word, unless the clock is reset, in which case only the constraint $0 < x < 1$ will be satisfied.

As a consequence, the algorithm is composed of two phases:

- reading the prefix of the word,
- reading the tail of the Zeno words.

A first important trick is the uniform stabilization. Indeed, all runs reading a same word stabilize from some point, but depending on the resets, this stabilization does not necessarily occur at the same point, and there may exist an unbounded number of such runs. Nevertheless, there exists a point after which all these runs are stabilized. The idea is roughly that at a fix point from which the sum of the future delays is smaller than 1, only a finite number of valuations are possible and the stabilization only depends on it.

Finally, to read the tail, the behavior of the automaton can be reduced in a natural way to that of a finite automaton. Hence, one can decide, given a finite set of states, whether all tails can be read and accepted from at least one of it. Then, we use the abstract transition system for one-clock timed automata from [OW05] to decide the existence of a set of states, reachable after a common finite timed word and from which some tails cannot be read. □

Conclusion

In this chapter we studied the emptiness and universality problems for timed automata with frequencies. The emptiness problem with frequency-based acceptance is decidable for timed automata with one clock and for forgetful aperiodic strongly non-Zeno timed automata, thanks to the computation of the bounds of the set of frequencies based on the corner-point abstraction. In the same way, if those timed automata are deterministic, one can decide the universality problem.

On the other hand, the universality problem in non-deterministic timed automata is harder. Our results about universality problems are summarized in the following table:

Universality	$\mathcal{L}_{\sharp\lambda}$	$\mathcal{L}_{>0}^{Zeno}$
one-clock	NPR	Decidable & NPR
several clocks	undecidable	undecidable

The undecidability of the general case comes from a reduction from the universality problem for finite timed words in timed automata with the usual semantics. The other consequence of this reduction is that the universality for one-clock timed automata is non-primitive recursive. The question of the decidability status remains open. However, we surprisingly proved that, for non-deterministic one-clock timed automata with positive frequency, the universality problem restricted to Zeno timed words is decidable but non-primitive recursive. Finally, an other open question is the decidability of the universality problem for forgetful timed automata with different semantics. We would need to carefully inspect the existing reduction [AD94] to find out whose widgets are not forgetful.

Conclusion

In this part, we introduced the notion of frequency of a run as the proportion of time elapsed in accepting locations. This permits to assign a value to each run, and thus to each timed word (taking the maximal value over the runs reading it). It can then be used in a language-theoretic approach to define quantitative languages associated with a timed automaton. One can then consider the set of timed words for which there is an execution of frequency greater than a threshold.

We developed techniques to compute the bounds of the set of frequencies in a timed automaton, with the motivation to decide in particular the emptiness of such languages. The framework is based on the corner-point abstraction introduced in [BBL08]. We first studied one-clock timed automata and developed techniques, but these do not extend to two-clock timed automata. Then, we proposed an extension to timed automata with several clocks whose cycles are forgetful, that is there are no convergences forced along them. Indeed, these phenomena make the frequencies harder to compute, more precisely, it weakens the link between frequencies of runs in a timed automaton and ratios of runs in its corner-point abstraction. For such timed automata, we proved that the set of frequencies is equal to the set of ratios (abstract frequencies) in the corner-point abstraction.

Beyond the context of frequency-based language, the central idea of this part was to explore the link between a timed automaton and its corner-point abstraction. The corner-point abstraction is a powerful tool allowing to preserve some information on timed behaviors. It was proved in [BBL08], that for all $\varepsilon > 0$, any run in the corner-point abstraction can be lifted to a run in the timed automaton preserving valuations up to ε. With the aim to preserve the frequency, we proved that this ε can be decreased along the run. We did it in such a way that the divergence of time allows to neglect these decreasing imprecisions in the computation of the frequency. Remark that we had to consider timed automata without convergence forced along cycles because such phenomena can force the imprecision ε to increase. To do so, the forgetfulness assumption allowed to establish technical lemmas which could be used in other contexts. Moreover this assumption, which is realistic from an implementability point of view, could be used to simplify other problems, or to identify a real behavior which for example does not satisfy a property. Indeed, one could consider as a realistic behavior a symbolic path iterating a forgetful cycle (equivalently, a path whose language has a positive entropy [BA11]). Finally, forgetfulness has already been used in the context of robust control of safety properties in timed automata in [SBMR13]. Forgetfulness is thus used to characterize timed automata which are robustly controllable.

Part V

Reachability of Communicating Timed Automata

Introduction

In this part, we are interested in the modeling of distributed systems where processes can communicate and in which timing constraints are important. In the untimed context, a fundamental model to represent systems of communicating processes, is the one of communicating finite-state machines [vB78]. Processes are then modeled by finite automata and they can exchange messages via communication channels. This model has been widely studied in the last decades. In this part, we propose to study a natural extension of communicating finite-state machines, where processes are modeled by timed automata in order to take into account timing constraints.

Communicating finite-state machines A communicating finite-state machine is simply a finite set of finite automata (called processes) which communicate via unbounded channels. More precisely, if there is a channel between a process p and a process q, then p can send messages in the channel and q can receive them. Channels are perfect, hence messages leave the channel only when q receives them. Channels are also FIFO (First In First Out): messages are received by q in the same order than they were sent by p. The graph whose vertices are the processes and edges are the channels is called the communication topology.

Communicating finite-state machines have the power of Turing machines [Pac82, BZ83]. As a consequence, one cannot decide the reachability problem: whether there exists an execution of the system which ends in an accepting state. Intuitively, cycles in the topology permit to encode the tape of a Turing machine. Moreover, if two processes are connected by two different paths of channels, the Post correspondence problem can also be encoded. The source can emit the tops of the blocks in one of the outgoing channels and the bottoms in the other channel, and the target process can receive messages two by two (one from each ingoing channel) to check whether they are similar. Intermediate processes along paths only forward messages.

Decidable topologies In fact, the reachability problem is undecidable if and only if there exists an undirected cycle in the topology [Pac82, BZ83]. We say that a topology without undirected cycle is a polyforest. Moreover, we say that a topology is decidable (resp. undecidable) if the reachability problem is decidable (resp. undecidable) in communicating finite-state machines with this topology.

Decidability comes from the fact that in polyforest topologies, considering channels of size one suffices for deciding reachability. Indeed, if there is a run ϱ reaching an accepting state, then there is another run ϱ' where each process does locally the same actions and which is eager, that is the length of the channels is bounded by one. Local runs of processes are interleaved following the simple rule: each message is received immediately after its emission. The run ϱ' is said to be a rescheduling of ϱ.

This rescheduling is possible only in polyforests. For example, consider two processes p and q which can communicate in both directions. Suppose there is a run where p sends as many a's desired and then receives as many b's as necessary, while q sends as many b's as wanted and then receives as

many a's as necessary. There is no possible rescheduling to obtain 1-bounded channels, because there is no interleaving of these two sequences of actions allowing to receive messages immediately after their emissions. In fact, it is possible only in polyforests because for each process, messages that are received from different channels, can be scheduled independently and cannot be constrained by the process itself as in the above example.

Other restrictions leading to decidability When channels are bounded, communicating finite-state machines have a finite number of states and the expressivity is the same as for finite automata. Yet, the boundedness of channels of communicating finite-state machines is undecidable. In order to obtain the decidability of reachability, several other restrictions have been considered. Let us give an overview of different approaches.

First of all, beyond the boundedness of channels, a weaker properties leads to decidability of the reachability problem: the existential boundedness. It states that there exist bounds over the channels such that every run admits a rescheduling of its actions which respects the bounds. Under this assumption, and knowing the bound, the reachability problem thus is decidable. It has also been shown that, roughly, the only case where the existential boundedness is decidable, is when the channel bound is known and the system of communicating finite-state machines is deadlock-free (*i.e.* there is no reachable configuration in which no transition is enable) [GKM07].

One-type message communicating finite-state machines have been explored in [PP92]. In this subclass, messages in a channel are all of the same type. This model is equivalent to Petri nets. If moreover it is required that the topology is cyclic, then the reachability problem is decidable. Weaker language restrictions for channels have been proposed in [MF85] generalized in [JJ93]. For these subclasses, the reachability problem is decidable, unfortunately the membership to these classes is undecidable. Nevertheless, a semi-algorithm for the reachability problem which terminates at least for the communicating finite-state machine of the defined subclass is proposed in [JJ93].

Another restriction of the communications, called half-duplex communication leads to decidability of the reachability problem [CF05]. More precisely, the reachability problem for two processes connected in both directions, assuming that at most one channel is non-empty in each configuration of the system along runs is decidable. The generalization of this assumption to any topology (*i.e.* assuming that at most one channel is non empty in each cycle) is not sufficient to obtain decidability. Nevertheless, a characterization of the decidable topologies with this assumption for communicating pushdown automata has been presented in [HLMS10].

Finally, an alternative model considering unreliable (or lossy) channels has also been investigated [CFI96]. Channels can then lose messages before their reception. With this assumption, the reachability problem is decidable whatever the topology. Recently, mixing of perfect and lossy channel has been studied in [CS08]. The reachability problem is decidable for the topology constituted of two processes connected by two channels, a perfect one and a lossy one. Nevertheless, if any channel is added to this simple topology, then the reachability problem becomes undecidable.

Timed distributed models Recently, there have been several works bringing time into models including concurrency or communications.

As discussed in the general introduction, timing aspects have been added to Petri nets to obtain several models: time Petri nets [Mer74], timed Petri nets [Ram74] or timed-arcs Petri nets [Han93].

We also presented the model of time-constrained message sequence charts graphs [IT11] which permits to represent concurrency and communications together with timing aspects, but in a higher level, focusing on communication behaviors. In particular, [CM06] proposes timed message sequence

charts as the semantics of communicating timed automata. Again relating message sequence charts and automata, communicating event-clock automata, a strict subclass of timed automata, are studied in [ABG07]. It is shown, among other results, that the reachability problem is decidable for arbitrary topologies over existentially-bounded channels.

On the other hand, ad hoc networks [DSZ10] are constituted of processes which can communicate by selective broadcast messages. Each process can communicate only with its neighbors. This model has been considered with both discrete and dense time in [ADR+11]. Several decidability and undecidability results are thus presented for different classes of timed ad hoc networks.

An extension of lossy channel systems including timing aspects has recently been presented in [AAC12]. The model considers messages together with their ages and receptions can happen only if the age of messages satisfies some guards. The reachability problem is then decidable for all topologies.

Models with loose synchronization over time elapsing have been studied in [IDP03] for discrete time and for dense time in [ABG+08]. In both approaches, local times of processes differ, but the communication policies are different. In [IDP03], messages can be exchanged via channels and the reachability problem is proved to be decidable for a restricted two-process topology. In [ABG+08], there are no messages, but the communication happens by observing the local clocks of the other processes. Several semantics are studied. In particular, the existence problem of an untimed word accepted for all drifts of clocks is undecidable, even for only two processes having one clock.

Urgency leads to undecidability Finally, the model closest to our setting is the communicating timed automata via perfect channels introduced in [KY06]. Indeed, in this part we propose to extend the results of Krcal and Yi [KY06], considering a slight extension of their model. Let us develop explanations about the contribution of [KY06]. The model considered is timed automata communicating via perfect FIFO-channels. Roughly, it is communicating finite-state machines where processes are modeled by timed automata instead of finite automata. Channels have the particularity to be urgent in the sense that, if a timed automaton can receive a message, then all internal actions (non-communication actions) are disabled. This model is extremely powerful. Considering sequences of processes where channels connect two successive processes (pipelines), it is proved that the reachability problem is decidable if and only if there are at most two processes. This result is very negative, but we are going to prove that this is due to the urgency of the channels. Relaxing this assumption, we obtain more positive results.

A first observation is that the proof in [KY06], which reduces the reachability problem in counter machines to the reachability problem in pipeline communicating timed automata, uses continuous time only in a discrete manner to synchronize actions of all processes. More precisely, at each time unit, each process does exactly one action (potentially internal). Counters are encoded by the number of some messages in the channels. Then, we come to a second observation which is that the zero test of counters is encoded thanks to the urgency of the receptions. As a consequence, the proof is based on the urgent semantics and relaxing this assumption makes the construction of the proof inapplicable. Nevertheless, the undecidability result is not only due to urgency, but also to the combination of synchronization (time) and urgency. In particular, urgency in communicating finite-state machines does not prevent the rescheduling to be performed in polyforest topologies. Hence, it is still possible to decide the reachability problem in such communicating finite-state machines considering only 1-bounded channels.

Contribution In this part, we consider communicating timed processes where a finite number of timed automata synchronize over the elapsing of time and communicate by exchanging messages over FIFO unbounded channels which can be urgent or not. We significantly extend the results of [KY06], by giving a complete characterization of the decidability frontier of reachability properties with respect to the communication topology when some channels are not urgent. Our study comprises both dense and discrete time.

- **Discrete time: Communicating tick automata** We provide a detailed analysis of communication in the discrete time model, where actions can only happen at integer time points. As a model of discrete time, we consider communicating tick automata, where the flow of time is represented by an explicit tick action. A process evolves from one time unit to the next one by performing a tick action, forcing all the other processes to perform a tick as well; all the other actions are asynchronous. This model of discrete time is called tick automata in [GHKK05], where they are considered with the Büchi semantics. A time-wrap lemma is formulated in a similar way as pumping lemmas for finite automata. It can thus be used to show that a discrete timed language is not regular. This interpretation of discrete time is related to the fictitious time of [AD94].

  As discussed above, the proof of [KY06] applies to discrete time. As a consequence, the reachability problem in pipeline communicating tick automata whose channels are urgent, is decidable if and only if there are at most two processes. In this part, we propose to study communicating tick automata with urgent channels as in [KY06], but also with non-urgent channels. In fact, we do not consider urgency directly, but we rather model it by introducing an additional emptiness test operation on channels on the side of the receiver. This allows us to discuss topologies where emptiness tests are restricted to certain components. We thus extend the results of [KY06] by providing a complete characterization of decidable topologies for communicating tick automata. We show that the reachability problem is decidable if, and only if, the topology is a polyforest (as for communicating finite-state machines), and, additionally, each weakly-connected component can test at most one channel for emptiness (*i.e.* along an undirected path of the topology, there is at most one channel whose emptiness is testable).

  Our results follow from topology-preserving mutual reductions between communicating tick automata and counter automata. As a consequence of the structure of our reductions, we show that channels and counters are mutually expressible, and similarly for emptiness tests and zero tests. This also allows us to obtain a complexity result for communicating tick automata. We show that reachability in a polyforest system of communicating tick automata over a topology without emptiness tests has is EXPSPACE-complete.

- **Dense time: Communicating timed automata** A first result is the complete characterization of the decidability frontier for communicating timed automata without urgency. We show that the reachability problem is decidable if, and only if, the communication topology is a polyforest. Thus, adding dense time does not change the decidability frontier compared to communicating finite-state machines and communicating tick automata. However, the complexity increases. From our results it follows that reachability in communicating timed automata is EXPSPACE-hard, and probably worst due to an exponential blow-up when translating from dense to discrete time. Nevertheless, the problem is in 2EXPSPACE.

  Then we expand the characterization considering two kinds of channels, depending on whether emptiness can be tested. We complete the undecidability picture for dense time, proving that a

topology with at least two channels whose emptiness is testable, in the same weakly connected component, is undecidable. All our results for dense time follow from a mutual, topology-preserving reduction to the discrete time model. Over polyforest topologies, we reduce from dense to discrete time when no channel can be tested for emptiness. Over arbitrary topologies, we reduce from discrete to dense time, even in the presence of emptiness tests. While the latter is immediate, the former is obtained via a rescheduling lemma for dense time automata which is interesting on its own, allowing us to schedule processes in fixed time-slots where send actions are always executed before receive actions. Unfortunately, this lemma does not immediately extend to channels whose emptiness is testable, and we do not have concrete ideas to get round the difficulty.

Outline Chapter 9 is devoted to the definitions of a uniform semantics for systems of communicating timed processes, together with the two main instantiations yielding the semantics of communicating timed automata and communicating tick automata. Then, the reductions between communicating tick automata and counter machines are presented in Chapter 10 with their consequences about decidability and complexity. Next, these results are partially translated to communicating timed automata thanks to mutual reductions with communicating tick automata. Finally, difficulties for the remaining open question is discussed.

Chapter 9

Communicating Timed Processes: a Uniform Semantics

In this chapter, we define communicating timed processes with various delay domains in a uniform way. We then instantiate the delay domain with $\mathbb{R}_+$ and $\{\tau\}$ to respectively obtain the semantics of communicating timed automata and communicating tick automata. Our model is very general, and in particular allows to test for emptiness of channels, which permits to model the urgency of [KY06]. In Section 9.3, we develop this link as well as consequences of the time synchronization of processes.

For readability, notations are sometimes different from the rest of the document but they are all defined or redefined here. For example, the definition of runs is a bit different because we focus on the reachability problem instead of languages problems.

9.1 Definition of communicating timed processes

In this section, we define communicating timed processes with discrete or continuous time thanks to a uniform semantics equipped with a delay domain. Processes actions are partitioned: there are communication actions, internal actions and delays (which synchronize all processes). In order to model urgency of [KY06], we introduce an extra action which tests the emptiness of a channel.

Definition 9.1 (Labeled transition system)**.** *A* labeled transition system *(LTS for short) is a tuple* $\mathsf{TS} = \langle S, S_I, S_F, A, \to \rangle$ *where* S *is a set of* states *with* initial states $S_I \subseteq S$ *and* final states $S_F \subseteq S$*,* A *is a set of* actions*, and* $\to \subseteq S \times A \times S$ *is a* labeled transition relation*.*

For simplicity, we write $s \xrightarrow{a} s'$ in place of $(s, a, s') \in \to$. A *path* in TS is an alternating sequence $\pi = s_0, a_1, s_1, \ldots, a_n, s_n$ of states $s_i \in S$ and actions $a_i \in A$ such that $s_{i-1} \xrightarrow{a_i} s_i$ for all $i \in \{1, \ldots, n\}$. We abuse notation and shortly denote π by $s_0 \xrightarrow{a_1 \cdots a_n} s_n$. The word $a_1 \cdots a_n \in A^*$ is called the *trace* of π. A *run* is a path starting in an initial state ($s_0 \in S_I$) and ending in a final state ($s_n \in S_F$).

We consider systems that are composed of several processes interacting with each other in two ways. Firstly, they implicitly synchronize over the passing of time. Secondly, they explicitly communicate through the asynchronous exchange of messages. For the first point, we represent delays by actions in a given *delay domain* $\mathbb{D}$. Typically, the delay domain is a set of non-negative numbers for dense time, or a finite set of abstract delays for discrete time.

Definition 9.2 (Timed processes)**.** *A* timed process *over the delay domain $\mathbb{D}$ is a labeled transition system $\mathsf{TS} = \langle S, S_I, S_F, A, \to \rangle$ such that $A \supseteq \mathbb{D}$ and* TS *satisfies the following condition:*

$$\forall s \in S_F, \forall d \in \mathbb{D}, \exists s' \in S_F : s \xrightarrow{d} s' \tag{9.1}$$

Actions in A are either synchronous delay actions *in $\mathbb{D}$, or* asynchronous actions *in $A \setminus \mathbb{D}$.*

Condition (9.1) simply means that a process which reaches an accepting state can let time elapse staying in accepting states. In particular it can let time elapse until the other processes also reach accepting states. Nevertheless, a transmission or a reception can lead from accepting states to non-accepting states. This assumption is quite natural, indeed a process which ended its execution do not have to block time elapsing for the other processes in practice.

Let us now introduce FIFO channels between processes as pairs (p, q) of processes, with the intended meaning that process p can send messages to process q.

Definition 9.3 (Topology)**.** *A communication* topology *is a triple $\mathcal{T} = \langle P, C, E \rangle$, where $\langle P, C \rangle$ is a directed graph comprising a finite set P of* processes *and a set of communication* channels *$C \subseteq P \times P$. Additionally, the set $E \subseteq C$ specifies channels that can be tested for emptiness.*

Let us define several notions around topologies. Channels in E are said to be *testable* channels. A topology $\langle P, C, E \rangle$ is said *test-free* if E is empty. A topology is *weakly-connected* if for any pair of processes, there is an undirected path of channels between them. A *weakly-connected component* $\mathcal{T}'$ of a topology $\mathcal{T}$ is a maximal weakly-connected subgraph of the $\mathcal{T}$, that is there exists no larger weakly-connected subgraph of $\mathcal{T}$ containing $\mathcal{T}'$. A topology is *acyclic* if it has no cycle. A topology is a *polyforest* if it has no undirected cycle. Moreover, a *polytree* topology is exactly a weakly-connected polyforest topology.

For a process $p \in P$, let $C[p] = C \cap (\{p\} \times P)$ be its set of outgoing channels, and let $C^{-1}[p] = C \cap (P \times \{p\})$ be its set of incoming channels. Processes may send messages to outgoing channels, receive messages from incoming channels, as well as test emptiness of incoming channels for testable channels.

Definition 9.4 (Communication actions)**.** *Let $\mathcal{T} = \langle P, C, E \rangle$ be a topology.*

- *Given a finite set M of messages, the set of possible* communication actions for process $p \in P$ *is* $A^p_{\mathrm{com}} = \{c!m \mid c \in C[p], m \in M\} \cup \{c?m \mid c \in C^{-1}[p], m \in M\} \cup \{c == \varepsilon \mid c \in E \cap C^{-1}[p]\}$.
- *The set of all* communication actions *is* $A_{\mathrm{com}} = \bigcup_{p \in P} A^p_{\mathrm{com}}$.

While send actions ($c!m$) and receive actions ($c?m$) are customary, we introduce the extra test action ($c == \varepsilon$) to model the *urgent semantics* of [KY06]. Actions not in $(\mathbb{D} \cup A_{\mathrm{com}})$ are called *internal actions*.

Definition 9.5 (Communicating timed processes)**.** *A* system of communicating timed processes *is a tuple $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, (\mathsf{TS}^p)_{p \in P} \rangle$ where $\mathcal{T} = \langle P, C, E \rangle$ is a topology, M is a finite set of* messages*, $\mathbb{D}$ is a delay domain, and, for each $p \in P$, $\mathsf{TS}^p = \langle S^p, S^p_I, S^p_F, A^p, \to^p \rangle$ is a timed process over $\mathbb{D}$ such that $A^p \cap A_{\mathrm{com}} = A^p_{\mathrm{com}}$.*

States $s^p \in S^p$ are called *local states* of p, while a *global state* $\vec{s} = (s^p)_{p \in P}$ is a tuple of local states in $\prod_{p \in P} S^p$. We give the semantics of a system of communicating timed processes in terms of a global labeled transition system. The contents of each channel is represented as a finite word over the alphabet M. Processes move asynchronously, except for delay actions that occur simultaneously.

Definition 9.6 (Semantics of communicating timed processes)**.** *The* semantics of a system $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, (\mathsf{TS}^p)_{p\in P}\rangle$ of communicating timed processes *is the labeled transition system* $[\![\mathcal{S}]\!] = \langle S, S_I, S_F, A, \to\rangle$ *where* $S = (\prod_{p\in P} S^p) \times (M^*)^C$, $S_I = (\prod_{p\in P} S^p_I) \times \{\varepsilon^C\}$, $S_F = (\prod_{p\in P} S^p_F) \times \{\varepsilon^C\}$, $A = \bigcup_{p\in P} A^p$, *and there is a transition* $(\vec{s}_1, w_1) \xrightarrow{a} (\vec{s}_2, w_2)$ *under the following restrictions:*

- *if* $a \in \mathbb{D}$, *then* $s_1^p \xrightarrow{a} s_2^p$ *for all* $p \in P$,
- *if* $a \notin \mathbb{D}$, *then* $s_1^p \xrightarrow{a} s_2^p$ *for some* $p \in P$, *and* $s_1^q = s_2^q$ *for all* $q \in P \setminus \{p\}$
  - *if* $a = c!m$, *then* $w_2(c) = w_1(c) \cdot m$ *and* $w_2(d) = w_1(d)$ *for all* $d \in C \setminus \{c\}$,
  - *if* $a = c?m$, *then* $m \cdot w_2(c) = w_1(c)$ *and* $w_2(d) = w_1(d)$ *for all* $d \in C \setminus \{c\}$,
  - *if* $a = (c == \varepsilon)$, *then* $w_1(c) = \varepsilon$ *and* $w_1 = w_2$, *and*
  - *if* $a \notin A_{\mathrm{com}}$, *then* $w_1 = w_2$.

To avoid confusion, states of $[\![\mathcal{S}]\!]$ will be called *configurations* in the remainder of the part. Given a path π in $[\![\mathcal{S}]\!]$, its *projection* to process p is the path $\pi|_p$ in TS^p obtained by projecting each transition of π to process p. In a transition $(\vec{s}_1, w_1) \xrightarrow{a} (\vec{s}_2, w_2)$ over a delay action $a \in \mathbb{D}$, all processes locally perform the transition. In this case, define its projection to process p to be the underlying transition $s_1^p \xrightarrow{a} s_2^p$. Otherwise, exactly one process p moves and the others stay put. Then , the projection to process q is $s_1^p \xrightarrow{a} s_2^p$ if $p = q$, and empty otherwise.

In the sequel, we study the reachability problem in communicating timed processes, which is defined as follows.

Definition 9.7.

- *The* reachability problem *asks, given a system of communicating timed processes* $\mathcal{S}$, *whether there exists a run in its semantics* $[\![\mathcal{S}]\!]$.
- *Two systems of communicating timed processes* $\mathcal{S}$ *and* $\mathcal{S}'$ *are said to be* equivalent *if* $[\![\mathcal{S}]\!]$ *has a run if and only if* $[\![\mathcal{S}']\!]$ *has a run.*

Recall that a run starts in initial states and ends in accepting states. We moreover require all channels to be empty at the end of a run, which simplifies our constructions later by guaranteeing that every sent message is eventually received. This is without loss of generality since configuration reachability and control-state reachability are easily inter-reducible.

9.2 Communicating timed or tick automata

In this section, we naturally use the uniform semantics of communicating timed processes to define communicating timed automata and communicating tick automata by instantiating the delay domain $\mathbb{D}$ respectively with $\mathbb{R}_+$ and $\{\tau\}$.

9.2.1 Communicating timed automata

Communicating timed automata are simply communicating timed processes synchronizing over the dense delay domain $\mathbb{D} = \mathbb{R}_+$. Let us define syntax and semantics of timed automata as timed processes before defining communicating timed automata.

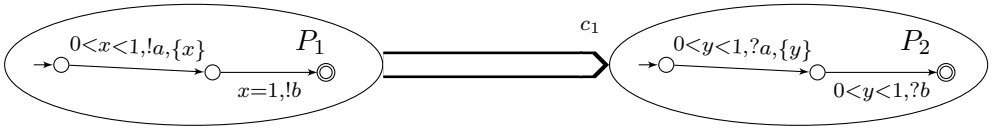


Figure 9.1: Example of a system of communicating timed automata $((P_i)_{1\leq i\leq 2}, \{c_1\})$.

Definition 9.8 (Timed automata)**.** *A* timed automaton $\mathcal{B} = \langle L, L_I, L_F, X, \Sigma, \Delta\rangle$ *is defined by a finite set of locations* L *with initial locations* $L_I \subseteq L$ *and final locations* $L_F \subseteq L$*, a finite set of* clocks X*, a finite alphabet* Σ *and a finite set* Δ *of transitions rules* $(\ell, \sigma, g, R, \ell')$ *where* $\ell, \ell' \in L$*,* $\sigma \in \Sigma$*, the guard* g *is a conjunction of constraints* $x\#c$ *for* $x \in X$*,* $\# \in \{<, \leq, =, \geq, >\}$ *and* $c \in \mathbb{N}$*, and* $R \subseteq X$ *is a set of clocks to reset.*

The semantics of timed automata can be given as timed processes over $\mathbb{D} = \mathbb{R}_+$.

Definition 9.9 (Semantics of timed automata)**.** *The semantics of* $\mathcal{B} = \langle L, L_I, L_F, X, \Sigma, \Delta\rangle$ *is given by the timed process* $[\![\mathcal{B}]\!] = \langle S, S_I, S_F, A, \to\rangle$ *with states* $S = L \times \mathbb{R}_+^X$*, initial states* $S_I = L_I \times \{0^X\}$*, final states* $S_F = L_F \times \mathbb{R}_+^X$*, actions* $A = \Sigma \cup \mathbb{R}_+$*, and transitions:*

- $(\ell, v) \xrightarrow{d} (\ell, v')$ *if* $d \in \mathbb{R}_+$ *and* $v'(x) = v(x) + d$ *for every clock* x*,*
- *and* $(\ell, v) \xrightarrow{\sigma} (\ell', v')$ *if there exists a rule* $(\ell, \sigma, g, R, \ell') \in \Delta$ *such that* g *is satisfied by* v *(defined in the natural way) and* $v' = v_{[R\leftarrow 0]}$*.*

We decorate a path $\pi = (\ell_0, v_0) \xrightarrow{a_0} (\ell_1, v_1) \xrightarrow{a_1} \cdots (\ell_n, v_n)$ in $[\![\mathcal{B}]\!]$ with additional *timestamps* $t_i = \sum\{a_j \mid j = 0, \ldots, i-1 \text{ and } a_j \in \mathbb{R}_+\}$: $\pi = (\ell_0, v_0) \xrightarrow{a_0, t_0} (\ell_1, v_1) \xrightarrow{a_1, t_1} \cdots (\ell_n, v_n)$. The t_i's give the date of occurrence of every transition.

We are now able to define communicating timed automata.

Definition 9.10 (Communicating timed automata)**.** *A* system of communicating timed automata *is a system of communicating timed processes* $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{R}_+, ([\![\mathcal{B}^p]\!])_{p\in P}\rangle$ *where for each* $p \in P$*,* $\mathcal{B}^p$ *is a timed automaton.*

Note that each timed automaton has access only to its local clocks. By Definition 9.5, each timed automaton performs communicating actions in A^p_{com} and synchronizes with all the other processes over delay actions in $\mathbb{R}_+$.

An example of a system of communicating timed automata is represented in Figure 9.1. There are two processes P_1 and P_2 and a channel $c_1 = (P_1, P_2)$. This is a pipeline of two processes. P_1 send an a and a b and P_2 receive them. To solve the reachability problem for this example, the question is thus whether there exist consistent timestamps, that is such that for every message, the reception is done after the emission. This system has some runs. For instance, P_1 can read the word $(!a, 0.5).(!b, 1.5)$ and P_2 can read $(?a, 0.7).(?b, 1.6)$. These local runs correspond to a run whose sequence of action is $(0.5).!a.(0.2).?a.(0.8).!b.(0.1).?b$. These timed words are compatible in the sense that receptions by P_2 are done after the corresponding emissions by P_1.

9.2.2 Communicating tick automata

Let us now give the definition of communicating tick automata which requires less notations.

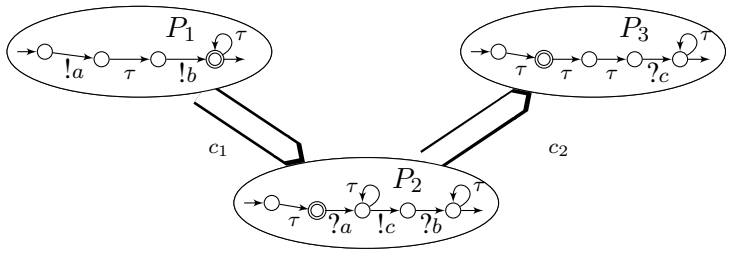


Figure 9.2: Example of a system of communicating tick automata $((P_i)_{1\leq i\leq 3}, \{c_1, c_2\})$.

Definition 9.11 (Communicating tick automata)**.** *A* system of communicating tick automata *is a system of communicating timed processes $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, (\mathsf{TS}^p)_{p\in P}\rangle$ such that $\mathbb{D} = \{\tau\}$ and for each $p \in P$, TS^p is a* tick automaton*, i.e., a timed process over $\mathbb{D}$ with finitely many states and actions.*

Thus, tick automata communicate with actions in A_{com} and, additionally, synchronize over the tick action τ. An example of a system of communicating tick automata $((P_i)_{1\leq i\leq 3}, \{c_1, c_2\})$ during an execution is represented in Figure 9.2. There are three processes P_1, P_2 and P_3 and two channels $c1 = (P_1, P_2)$ and $c2 = (P_2, P_3)$, hence the topology is a pipeline of three processes. Current states of the processes are represented by double circles. An execution of the system necessarily starts with an emission of a by P_1 and then the three processes synchronize on a τ-transition. Then, in the current execution P_1 emitted a b, but a could have been received by P_2 before. The current execution can be prolongated in a run with the sequence of actions: $!a.\tau.!b.?a.\tau.\tau.!c.?b.?c$. Condition 9.1 implies that from accepting states of tick automata, there is a τ-transition to an accepting state. When there is a single accepting state in the processes, there necessarily is a loop labeled by τ over them. In the sequel, we sometimes omit these loops.

The global synchronization over τ-transitions makes communicating tick automata more expressive than communicating finite-state machines, in the sense that ticks can forbid re-orderings of communication actions that are legitimate without ticks. Notice that there is only one tick symbol in $\mathbb{D}$. With two tick symbols, reachability is already undecidable for the one channel topology $p \to q$ without emptiness test. We give a proof of this result in section 9.3.3.

9.3 Discussion about the models

In this section, we first discuss the key notion of emptiness test to model urgency. Then, we illustrate, in communicating timed processes, the impossibility to re-order runs in order to bound the size of channels in the same way as for communicating finite-state machines. Finally, we detail the undecidability result about communicating tick automata if two different ticks are allowed.

9.3.1 Modeling urgency with emptiness test

In the *urgent semantics* for receive actions of [KY06], if a message can be received by a process, then internal actions are disabled (while other communication and delay actions are still enabled). In our model, instead of defining a separate urgent semantics, we introduce the extra test action $c == \varepsilon$, which allows us to locate more precisely where in the topology is the urgent semantics (i.e., test action) used. Below, we show how to implement the urgent semantics of [KY06] with the test action.

We need to ensure that internal actions of control states where also a receive action $c?m$ is available can be executed only if m cannot be received from c. This can happen if either (1) c is empty, or

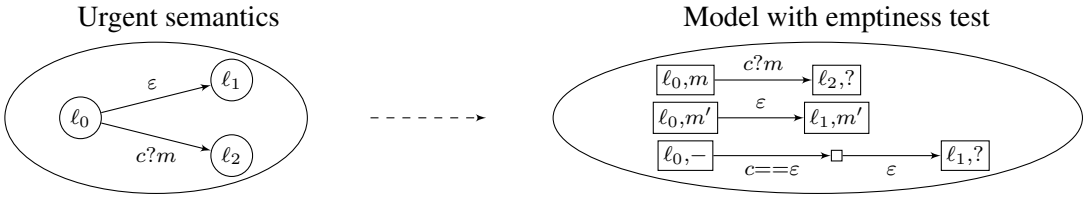
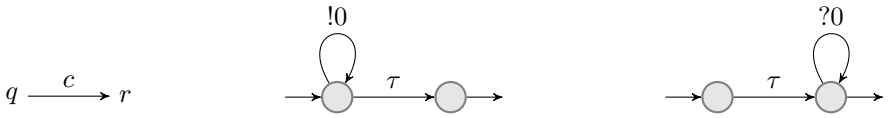


Figure 9.3: Illustration of the construction to model urgency of the reception of m.



(a) Topology

(b) Tick automaton for process q

(c) Tick automaton for process r

Figure 9.4: A simple system of communicating tick automata that is *not* existentially-bounded.

(2) it is not empty and the message in front of the channel is $m_s \neq m$. Let $M(\ell) = \{m \mid \ell \xrightarrow{c?m} \ell'\}$ be the set of messages that can be read from a given control location ℓ for a fixed channel c. For (2), we modify the automaton with a standard construction to store into its finite control the first message m_s that can be received from c (if any), and check that $m_s \notin M(\ell)$ before the internal action can be executed. For (1), in the case no message m_s is stored in the location, the internal action is preceded by a test action $c == \varepsilon$ (by introducing an intermediate state).

The construction is illustrated in Figure 9.3. More precisely, it represents how to translate the simple case where from a location ℓ in a timed automaton with urgent semantics, one can either perform an internal action if the reception is not enabled, or receive a message m. There is three possible locations in the translation.

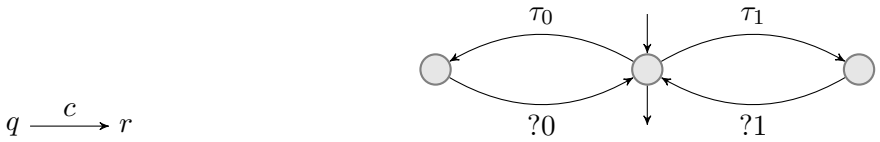
- the first message in the channel is m, then the ε-edge is disabled;
- the first message in the channel is $m' \neq m$, then the reception of m is disabled;
- there is no message in the buffer, then the reception of m is disabled and we check that the channel is empty before performing the internal action.

The target states of the form $(\ell, ?)$ mean that there is a copy of this state (and of the ingoing edge) for each possible message in front of the channel instead of ?, and one with no message. We, in some sense, guess the future reception or the emptiness of the channel.

9.3.2 On the power of time

Consider the topology with two processes q and r and a channel from q to r (that cannot be tested for emptiness). Formally, this topology is the triple $\mathcal{T} = \langle \{q, r\}, \{(q, r)\}, \emptyset \rangle$. It is known that every communicating finite-state machine with topology $\mathcal{T}$ is existentially 1-bounded, i.e., each run can be re-ordered into a run where the channel always contains at most one message [Pac82, HLMS10]. However, this property doesn't hold for systems of communicating tick automata.

Consider the example depicted in Figure 9.4. Because of the global synchronization enforced by the tick action τ, the first reception necessarily occurs after the last transmission. Hence, this example



(a) Topology (b) Multi-tick automaton for process r

Figure 9.5: Simulation of a perfect channel automaton by a 2 tick automaton.

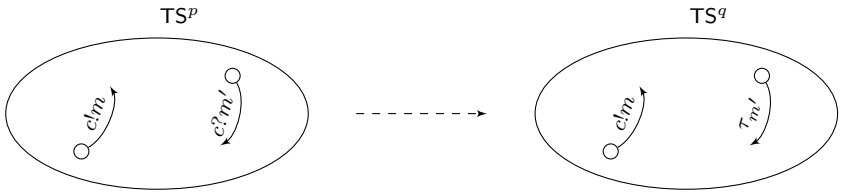


Figure 9.6: Illustration of the simulation of the communication actions of TS^p in TS^q.

is not existentially-bounded: for every bound $B \in \mathbb{N}$, there exists a run with no B-bounded reordering. This shows that systems of communicating tick automata are more expressive than CFSM. Alternatively, from a language viewpoint, the trace language of this example is $\{(!0)^n\tau(?0)^n \mid n \in \mathbb{N}\}$. However, no CFSM (with topology $\mathcal{T}$) has the same trace language (where τ would be an internal action). Note that a similar example can naturally be constructed for communicating timed automata by putting ε-transitions instead of τ-transitions with guards $x = 1$ and $y = 1$ in the respective processes and never resetting them.

9.3.3 Undecidability of multi-tick automata

One could consider a more expressive model where communicating tick automata can synchronize over a two distinct tick actions $\{\tau_0, \tau_1\}$, instead of just one tick τ. However, in the simplest non-trivial topology $\mathcal{T} = \langle\{q, r\}, \{(q, r)\}, \emptyset\rangle$ with two processes q, r and a channel from q to r with no emptiness tests, reachability becomes undecidable already with two tick actions. In fact, a perfect channel automaton $\mathcal{S} = \langle\langle\{p\}, \{(p, p)\}, \emptyset\rangle, M, \emptyset, (\mathsf{TS}^p)\rangle$ (for which reachability is undecidable [BZ83]) can be simulated by topology $\mathcal{T}$ above. Without loss of generality, assume $M = \{0, 1\}$. $\mathcal{S}$ can be simulated by a system of two communicating finite-state automata $\mathcal{S}' = \langle\mathcal{T}, M, \mathbb{D}, (\mathsf{TS}^q, \mathsf{TS}^r)\rangle$ over topology $\mathcal{T} = \langle\{q, r\}, \{(q, r)\}, \emptyset\rangle$, and where $\mathbb{D} = \{\tau_0, \tau_1\}$, TS^r is shown in Figure 9.5, and TS^q is defined as follows.

Let c be the channel (q, r). Figure 9.6 illustrates the principle of the simulation of the communication actions of TS^p in TS^q. The send actions $!m$ of p are seamlessly performed by q as $c!m$. Since q (unlike p) cannot directly read from the channel (only r can), to simulate a receive action $?m$ of p, q performs the corresponding tick action τ_m in order to force process r to read message m on its behalf. As a consequence, we derive the following theorem.

Theorem 9.1. *Let $\mathcal{T}$ be a topology with at least one channel. Then, the reachability problem for communicating multi-tick automata with at least two distinct tick actions and with topology $\mathcal{T}$ is undecidable.*

Conclusion

We defined communicating timed processes with perfect channels whose emptiness is potentially testable. This general model allows in particular to define the semantics of systems of communicating timed or tick automata. The testability of some channels allows one to model the urgency of [KY06]. In the sequel of the part, we extend the result of [KY06] considering also channels whose emptiness is not testable. The proof of undecidability in [KY06] uses time only to synchronize processes in such a manner that they all do exactly one action at each time unit. As a consequence of this observation, we first work on the simpler model of communicating tick automata which also permits this synchronization.

Chapter 10

Reachability Problem in Communicating Tick Automata

Introduction

As presented in the introduction of this part, the reachability problem in communicating finite-state machines with a fixed topology is decidable if and only if this topology is a polyforest [Pac82, BZ83]. In this case, the problem is PSPACE-complete and it is sufficient to consider 1-bounded channels. As discussed in Section 9.3.2, the rescheduling trick to consider bounded channels does not work when there is synchronization on time (discrete or dense).

In this chapter, we study decidability and complexity of communicating tick automata. Our main technical tool consists of mutual reductions to/from counter automata, showing that, in the presence of tick actions:

- each channel is equivalent to a counter, and
- each emptiness test on a channel is equivalent to a zero test on the corresponding counter.

This allows us to derive a complete characterization of decidable topologies, and also to obtain complexity results. We start by defining communicating counter automata.

10.1 Communicating counter automata

Let us briefly fix the notations for counter automata. Operations on a counter x are $x{+}{+}$ (increment), $x{-}{-}$ (decrement) and $x{=}{=}0$ (zero test). Given a set X of counters, we write $\mathtt{Op}(X)$ for the set of operations over counters in X.

Definition 10.1 (Counter automata)**.** *A* counter automaton *is a classical Minsky machine [Min67] $\mathcal{S}' = \langle L, L_I, L_F, A, X, \Delta\rangle$ with finitely many locations L, initial locations $L_I \subseteq L$, final locations $L_F \subseteq L$, a finite set of non-negative counters X, alphabet of actions $A \supseteq \mathtt{Op}(X)$, and transition rules $\Delta \subseteq L \times A \times L$.*

As usual, the semantics is given as a labeled transition system $[\![\mathcal{S}']\!] = \langle S, S_I, S_F, A, \to\rangle$ where $S = L \times \mathbb{N}^X$, $S_I = L_I \times \{0^X\}$, $S_F = L_F \times \{0^X\}$, and the transition relation $\to$ is the smallest relation such that: for all $(\ell, a, \ell') \in \Delta$ and for all $x \in X$,

- if $a = (x{+}{+})$ then, for all $v, v' \in \mathbb{N}^X$ such that $v'(x) = v(x) + 1$ and for all $y \in X \setminus \{x\}$ $v(y) = v'(y)$, $(\ell, \mathsf{v}) \xrightarrow{a} (\ell', \mathsf{v}')$;
- if $a = (x{-}{-})$ then, for all $v, v' \in \mathbb{N}^X$ such that $v'(x) = v(x) - 1$ and for all $y \in X \setminus \{x\}$ $v(y) = v'(y)$, $(\ell, \mathsf{v}) \xrightarrow{a} (\ell', \mathsf{v}')$;
- if $a = (x{=}{=}0)$ then, for all $v, v' \in \mathbb{N}^X$ such that $v'(x) = v(x) = 0$ and $v = v'$, $(\ell, \mathsf{v}) \xrightarrow{a} (\ell', \mathsf{v}')$.

For technical reason, we assume that acceptance is with zero counters. This is without loss of generality because from a counter machine without this assumption, one can build an equivalent machine whose reachability problem with zero counters is equivalent to the reachability problem for the original machine. The construction is simple and consists in adding an accepting sink location reachable from the other accepting locations by decreasing counters and in which one can decrease all the counters.

Definition 10.2 (Communicating counter automata)**.** *A* system of communicating counter automata *is a system of communicating timed processes $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, (\llbracket S^p \rrbracket)_{p \in P} \rangle$ such that $\mathbb{D} = \emptyset$ and each S^p is a counter automaton.*

By Definition 9.5, this entails that each counter automaton performs communication actions in A^p_{com}. Notice that, since the delay domain is empty, no synchronization over delay action is possible.

10.2 From tick automata to counter automata.

In this section, we present an intuition for the reduction from tick automata to counter automata. We first present a reduction from communicating tick automata with a given topology to communicating counter automata with the same topology. Then, rescheduling runs of the counter machine, we explain that the reduction can be done from communicating tick automata with polyforest topologies to a product of non-communicating counter automata. The goal of this reduction is to derive decidability and complexity results for the reachability problem in communicating tick automata.

Proposition 10.1. *Let $\mathcal{T}$ be a topology. For every system of communicating tick automata $\mathcal{S}$ with topology $\mathcal{T}$, we can build, in polynomial time, an equivalent system of communicating counter automata $\mathcal{S}'$ with the same topology.*

Sketch of the proof. Let us informally explain the principle of the construction. The complete proof is mainly due to my coauthors Lorenzo Clemente, Frédéric Herbreteau and Grégoire Sutre. It can be found in the research report [CHSS12].

Let $\mathcal{S}$ be a system of communicating tick automata over a polytree topology. We build a system of communicating counter automata $\mathcal{S}'$ over the same topology, which is equivalent with respect to the reachability problem. Synchronization on delay actions is not possible in communicating counter automata. Intuitively, we implement synchronization on the delay action τ in $\mathcal{S}$ by communication in $\mathcal{S}'$ Let us describe the outline of the reasoning.

- **A new message called τ.** We introduce a new type of message, also called τ, which is broadcast by all processes in $\mathcal{S}'$ each time there is a synchronizing tick action in $\mathcal{S}$.
- **A fair desynchronization.** Since communication through channels is by nature asynchronous, we allow the sender and the receiver to be momentarily desynchronized during the computation. However, we impose the desynchronization to be asymmetric. The receiver is allowed to be

"ahead" of the sender (with respect to the number of ticks performed), but never the other way around. This ensures causality between transmissions and receptions, by forbidding that a message is received before it is sent.

- **Definition of counters.** To keep track of the exact amount of desynchronization between sender and receiver (as the difference in number of ticks), we introduce counters in $\mathcal{S}'$. We endow each process p with a non-negative counter $\mathtt{x}_c^p$ for each channel $c \in C^{-1}[p]$ from which p is allowed to receive. The value of counter $\mathtt{x}_c^p$ measures the difference in number of ticks τ between p and the corresponding sender along c.

- **Increments and decrements.** Whenever a process p performs a synchronizing tick action τ in $\mathcal{S}$, in $\mathcal{S}'$ it sends a message τ in broadcast onto all outgoing channels; at the same time, all its counters $\mathtt{x}_c^p$ are incremented, recording that p, as a receiver process, is one more step ahead of its senders. When one such τ-message is received by a process q in $\mathcal{S}'$ along channel c, the corresponding counter $\mathtt{x}_c^q$ is decremented; similarly, this records that the receiver process along c is getting one step closer to the sender process p.

- **Correctness of the emptiness test.** While proper ordering of receptions and transmissions is ensured by the fact counters are non-negative, testing emptiness of the channels is more difficult. In fact, a receiver, which in general is ahead of the sender, might see the channel as empty at one point (thus the test is positive), but then the sender might later (*i.e.* after performing some tick) send some message, and the earlier test should actually have failed in the synchronized system $\mathcal{S}$ (false positive). We avoid this difficulty by imposing that the counter $\mathtt{x}_c^q$ is equal to zero when a test action is simulated. Thus enforcing that the receiver q is synchronized with the corresponding sender along channel c on emptiness tests, which ensures that the result of the test is sure. To do so, we simply add a zero test $\mathtt{x}_c^q$==0 to the test action $c == \varepsilon$ by q.

- **Resynchronization.** Given an accepting run in the system of communicating counter automata $\mathcal{S}'$, counters could be non zero, whereas processes, in a run of $\mathcal{S}$, have to perform the same number of τ actions. In a similar way, the numbers of τ actions in each component can be different in the components of $\mathcal{S}'$, whereas they have to be equals in $\mathcal{S}$. In fact, thanks to Condition (9.1), if there is an accepting run in $\mathcal{S}'$, the corresponding partial run in $\mathcal{S}$, which ends in accepting locations, can be prolongated in a run of $\mathcal{S}$ by simply adding the suitable number of τ's at the end, in each process, staying in accepting states. □

Remark 10.1. *There are as many counters in the built communicating counter machine for each process as channels in the same process of the original system of communicating tick automata. Moreover, the number of counters which have to be zero testable also corresponds to the number of testable channels.*

It has been shown that, on polytrees, runs of communicating processes (even infinite-state) can be rescheduled to satisfy the so-called *eagerness* requirement, where each transmission is immediately followed by the matching reception [HLMS10]. Their argument holds also in the presence of emptiness tests, since an eager run cannot disable $c == \varepsilon$ transitions. Indeed, making receptions closer to emissions can only empty the channels more often. Thus, by restricting to eager runs, communication behaves just as a rendezvous synchronization.

Then, one can reduce the reachability problem in polytrees system of communicating tick automata to the reachability problem in non-communicating counter automata. Indeed, given a polytree

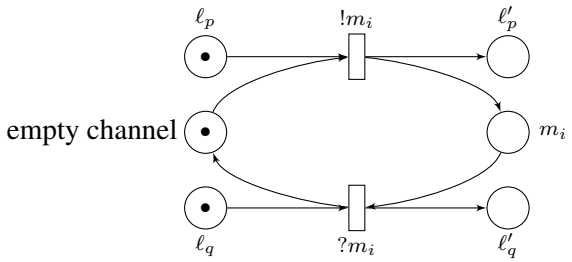


Figure 10.1: Encoding of the synchronization.

system $\mathcal{S}$ of communicating tick automata, one can then obtain an equivalent counter automaton to $\mathcal{S}$ by taking the product of all process counter automata synchronizing over τ's and emissions-receptions of the same message. Moreover, this reduction applies to polyforests. First, one can apply the reduction of Proposition 10.1 to each weakly-connected component. Then, thanks to Condition (9.1), numbers of τ actions performed in each component, can be made uniform, staying in accepting locations. Hence, there is a run in the polyforest if and only if there are runs in all the components which are polytrees. As a consequence, we obtain the following theorem.

Theorem 10.1. *For every polyforest topology $\mathcal{T}$, the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$ is reducible, in polynomial time, to the reachability problem for products of (non-communicating) counter automata.*

Remark 10.2. *By Remark 10.1, here again, there are as many channels (resp. testable channels) in the system of communicating tick automata as counters (resp. zero testable counters) in the built counter automaton.*

When no test is allowed, we obtain the complexity for the reachability problem in polyforest topologies. Indeed, it has the same complexity as the coverability problem for Petri nets, which is known to be EXPSPACE-complete [Lip76, Rac78].

Corollary 10.1. *The reachability problem for systems of communicating tick automata with test-free polyforest topologies is EXPSPACE-complete.*

Proof. The idea of the proof is that the reachability problem in our products of counter automata can be reduced, in polynomial time, to the coverability problem in Petri nets.

Let $\mathcal{S}$ be a polyforest system of communicating tick automata, and let $\mathcal{S}'$ be the equivalent product of counter automata obtain by Theorem 10.1. Each counter automaton $\mathcal{C}$ can be naturally transformed into a Petri net, by translating each location of $\mathcal{C}$ in a place and each edge of $\mathcal{C}$ in a transition, and defining the initial marking as the marking where there only is one token in the place corresponding to the initial location. Then, the synchronization over emissions and receptions can be encoded by additional places and transitions. Each one-bounded channel is encoded by $|M|+1$ places where $M = \{m_1, \cdots, m_{|M|}\}$ is the message alphabet. These places respectively represent the empty channel and the channel containing the message m_i. The Petri net is built in such a way that there always is exactly one token in only one of these places. The principle of the encoding is represented in Figure 10.1. A transition, which corresponds to an emission of the message m_i in a channel, can be fired only if the channel is empty, and when it is fired, the token in the place encoding the emptiness of the channel is placed in the place encoding that there is the message m_i in the channel. On the other hand, a

transition, which corresponds to a reception of the message m_i in a channel, can be fired only if the channel contains this message, and when it is fired, the token in the place encoding that there is the message m_i in the channel is placed in the place encoding the emptiness of the channel.

The size of the resulting Petri net is polynomial in the size of $\mathcal{S}'$, and hence in the size of $\mathcal{S}$. Moreover, a marking where there tokens encoding the current locations are in places corresponding to accepting locations is coverable if and only if there exists a run in $\mathcal{S}'$, and hence if and only if there exists a run in $\mathcal{S}$. □

10.3 From counter automata to tick automata

In the previous section, we presented a simulation of polyforest systems of communicating tick automata by counter automata without communication. In this section we propose a reciprocal reduction. More precisely, we reduce the reachability problem for non-communicating counter automata to the reachability problem for systems of communicating tick automata with star topology, that is topology where there is a central process p such that all the channels have p either as sender or as receiver and not both (see Figure 10.2 for a scheme).

Definition 10.3 (Star topology)**.** *A topology $\mathcal{T} = \langle P, C, E\rangle$ is called a* star topology *if there exist two disjoint subsets Q, R of P and a process p in $P \setminus (Q \cup R)$ such that $P = \{p\} \cup Q \cup R$ and $C = (R \times \{p\}) \cup (\{p\} \times Q)$.*

Then, the following theorem holds.

Theorem 10.2. *Let $\mathcal{T}$ be a star topology with α channels, of which β can be tested for emptiness. The reachability problem for (non-communicating) counter automata with α counters, of which β can be tested for zero, is reducible, in linear time, to the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$.*

The idea is to simulate each counter with a dedicated channel, thus the number of counters is the number of channels in $\mathcal{T}$. Moreover, our reduction is uniform in the sense that it works independently of the exact arrangement of channels in $\mathcal{T}$, which we take not to be under our control. Without loss of generality, we consider counter automata where all actions are counter operations (i.e., $\Delta \subseteq L \times \mathtt{Op}(X) \times L$).

Sketch of the proof of Proposition 10.1. This result is mainly due to my coauthors Lorenzo Clemente, Frédéric Herbreteau and Grégoire Sutre. As a consequence, I only give the intuition of the proof which can be found in its entirety in the research report [CHSS12].

Let us first consider an arbitrary star topology $\mathcal{T} = \langle P, C, E\rangle$ with the following set of processes $P = \{\mathtt{p}\} \cup Q \cup R$ where $Q = \{\mathtt{q}_1, \ldots, \mathtt{q}_m\}$, $R = \{\mathtt{r}_1, \ldots, \mathtt{r}_n\}$, $m, n \in \mathbb{N}$, and set of channels $C = \{\mathtt{p}\} \times Q \cup R \times \{\mathtt{p}\}$ and in which emptiness of all channels can be tested (*i.e.* $E = C$). This topology is depicted in Figure 10.2 (bottom left). Let $\mathcal{S}'$ be a counter automaton with $m + n$ counters, namely $X = \{\mathtt{x}_1, \ldots, \mathtt{x}_m\}$ and $Y = \{\mathtt{y}_1, \ldots, \mathtt{y}_n\}$. The counters are split into X and Y in arbitrary way to reflect the star topology $\mathcal{T}$, which is given a priori. We build, from $\mathcal{S}'$, an equivalent system of communicating tick automata $\mathcal{S}$ with topology $\mathcal{T}$. Let us informally explain how the reduction works.

- **Outline.** The process $\mathtt{p}$ simulates the control-flow graph of the counter automaton, and the counters $\mathtt{x}_i$ and $\mathtt{y}_j$ are respectively simulated by the channels $(\mathtt{p}, \mathtt{q}_i)$ called $\mathtt{c}_i$ and channels $\mathtt{r}_j, \mathtt{p})$ called $\mathtt{d}_j$.

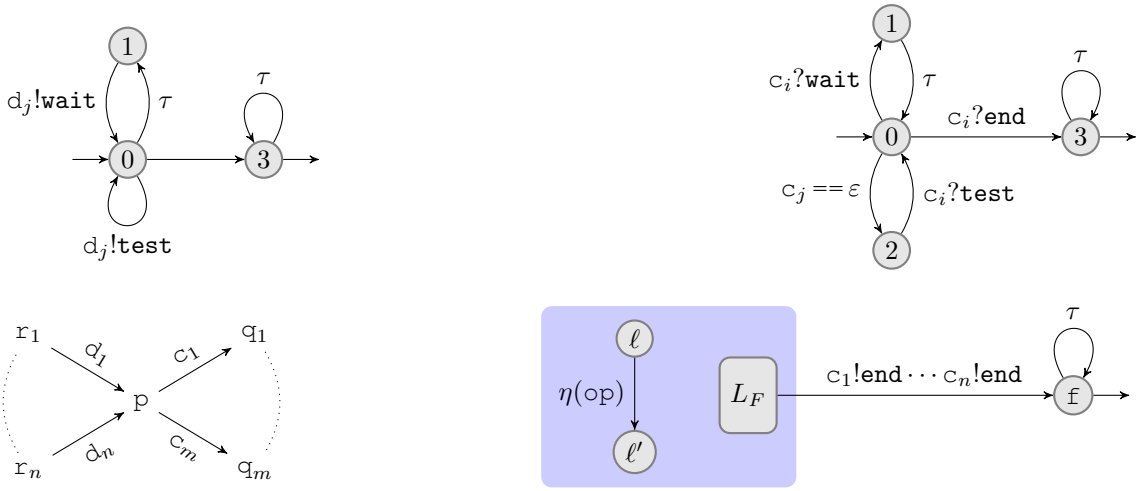


Figure 10.2: Simulation of a counter automaton by a system of communicating tick automata: Tick automata for $\mathtt{r}_j$ (top left) and $\mathtt{q}_i$ (top right), topology (bottom left), tick automaton for $\mathtt{p}$ (bottom right).

- **Messages.** In order to define $\mathcal{S}$, we need to provide its message alphabet and one tick automaton, for each process p in P. The message alphabet is $M = \{\mathtt{wait}, \mathtt{test}\}$. Actions performed by processes in P are either communication actions or the delay action τ.
- **Loose synchronization.** Processes $\mathtt{r}_j$'s are assigned the tick automaton of Figure 10.2 (top left), and processes $\mathtt{q}_i$'s are assigned the tick automaton of Figure 10.2 (top right). Intuitively, communications on `wait` messages are loosely synchronized using the τ actions in $\mathtt{q}_i$ and $\mathtt{r}_j$, so that $\mathtt{p}$ can control the rate of their reception and transmission.
- **Role of process $\mathtt{p}$.** The process $\mathtt{p}$ is a larger tick automaton than $\mathtt{q}_i$'s and $\mathtt{r}_j$'s. It simulates control states of $\mathcal{S}'$ and communicates with other processes for the simulation of counter operations. It is roughly represented on Figure 10.2 (bottom right). It contains roughly the same control states as in $\mathcal{S}'$ which are connected by the same counter operations. These operations can require several transitions in $\mathtt{p}$ to be simulated, in this case, some intermediate control states are added. The simulation preserves the control-flow graph of $\mathcal{S}'$. Hence, we simply explain how to translate counter operations of $\mathcal{S}'$ into communication actions and τ actions.
  - **Encoding of the counters.** The number of `wait` messages in channels $\mathtt{c}_i$ and $\mathtt{d}_j$ respectively encodes the value of counters $\mathtt{x}_i$ and $\mathtt{y}_j$.
  - **Increment of $\mathbf{x}_i$ / decrement of $\mathbf{y}_j$.** Incrementing $\mathtt{x}_i$ amounts to sending `wait` in $\mathtt{c}_i$, and decrementing $\mathtt{y}_j$ amounts to receiving `wait` from $\mathtt{d}_j$. Both actions can be performed by $\mathtt{p}$.
  - **Decrement of $\mathbf{x}_i$ / increment of $\mathbf{y}_j$.** Decrementing $\mathtt{x}_i$ is more involved, since $\mathtt{p}$ cannot receive from the channel $\mathtt{c}_i$. Instead, $\mathtt{p}$ performs a τ action in order to force a τ action in $\mathtt{q}_i$, hence, a receive of `wait` by $\mathtt{q}_i$. But all other processes also perform the τ action, so $\mathtt{p}$ compensates in order to preserve the number of `wait` messages in the other channels. To do so, process $\mathtt{p}$ simply receives `wait` from all channels $\mathtt{d}_j$'s and sends `wait` in all channels $\mathtt{c}_h$'s such that $h \neq i$. The simulation of the increment of $\mathtt{y}_j$ can similarly be done.

– **Zero test.** When $\mathtt{p}$ simulates $\mathtt{x}_i$==0, it simply sends `test` in the channel $\mathtt{c}_i$. This message is eventually received by $\mathtt{q}_i$ since all channels must be empty at the end of the simulation. The construction guarantees that the first receive action of $\mathtt{q}_i$ after the send action $\mathtt{c}_i$!`test` of $\mathtt{p}$ is the matching receive $\mathtt{c}_i$?`test`. This means, in particular, that the channel is empty when $\mathtt{p}$ sends `test` in $\mathtt{c}_i$. The same device is used to simulate a zero test of $\mathtt{y}_j$, except that the roles of $\mathtt{p}$ and its peer (here, $\mathtt{r}_j$) are reversed. Clearly, channels that need to be tested for emptiness are those encoding counters that are tested for zero.

– **End of the execution.** In their final state, $\mathtt{q}_i$'s and $\mathtt{r}_j$'s do nothing except τ actions, as required by Condition (9.1). Processes $\mathtt{r}_j$'s can move freely to its final control state, then, in order not to accept to fast a run, $\mathtt{q}_i$'s must receive an `end` message from $\mathtt{p}$ to move in a final control state. Indeed, without this trick, a process $\mathtt{q}_i$ could stop the simulation of counter $\mathtt{y}_i$ by staying in a final control state and performing τ's without receiving messages. As a consequence, from control states of $\mathtt{p}$, corresponding to final control states of $\mathcal{S}'$, there is a sequence of send of `end` messages in all the c_i's. □

10.4 Characterization of the decidable topologies

Since our mutual reductions between counter machines and polyforest systems of communicating tick automata show how transform counters into channels, and zero tests into emptiness tests, we may completely characterize which topologies have a decidable reachability problem, depending on exactly which channels can be tested for emptiness. Intuitively, decidability holds even in the presence of multiple emptiness tests, provided that each test appears in a different weakly-connected component.

Theorem 10.3 (Characterization of decidable topologies)**.** *Given a topology $\mathcal{T}$, the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$ is decidable if and only if $\mathcal{T}$ is a polyforest containing at most one testable channel in each weakly-connected component.*

Proof. Let us prove both implications of this theorem.

- For one direction, assume that the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$ is decidable. The topology $\mathcal{T}$ is necessarily a polyforest, since the reachability problem is undecidable for non-polyforest topologies even without ticks [Pac82, BZ83]. Suppose that $\mathcal{T}$ contains a weakly-connected component with (at least) two channels that can be tested for emptiness. By an immediate extension of Theorem 10.2 to account for the undirected path between these two channels, we can reduce the reachability problem for two-counter automata to the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$. Since the former is undecidable, each weakly-connected component in $\mathcal{T}$ contains at most one testable channel.

- For the other direction, assume that $\mathcal{T}$ is a polyforest with at most one testable channel in each weakly-connected component, and let $\mathcal{S}$ be a system of communicating tick automata with topology $\mathcal{T}$. Thus, $\mathcal{S}$ can be decomposed into a disjoint union of independent systems $\mathcal{S}_1, \ldots, \mathcal{S}_n$, where each $\mathcal{S}_i$ has a polytree topology $\mathcal{T}_i$ containing at most one testable channel. The only interactions between $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are through τ actions. But these interactions are superficial: every run of $\mathcal{S}_i$ can be continued with arbitrarily many τ actions, as required by

(9.1). Therefore, $\mathcal{S}$ has a run if, and only if, each $\mathcal{S}_i$ has a run. By Theorem 10.1, the reachability problem for systems of communicating tick automata with topology $\mathcal{T}_i$ is reducible to the reachability problem for counter automata where only one counter can be tested for zero. As the latter is decidable [Rei08, Bon11], the former is decidable, too. We obtain that the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$ is decidable. □

Conclusion

Even though global synchronization makes communicating tick automata more expressive than communicating finite-state machines, our characterization shows that the reachability problem is decidable for exactly the same topologies (that is, polyforests). However, while reachability problem for communicating finite-state machines is PSPACE-complete, it is EXPSPACE-complete for communicating tick automata. In the next chapter, we extend some results to communicating timed automata, thanks to technical lemmas allowing one to reschedule runs, and thus allowing one to discretize continuous timed systems.

Chapter 11

Reachability Problem in Communicating Timed Automata

Introduction

In this chapter, we consider communicating timed automata, that is communicating timed processes synchronizing over the dense delay domain $\mathbb{D} = \mathbb{R}_+$. We extend results for tick automata of Chapter 10 to the case of timed automata. To this end, we present mutual, topology-preserving reductions between communicating tick automata and communicating timed automata.

The idea is to abstract each timed automaton by a tick automaton preserving essential informations about the synchronization with respect to time. First, each timed automaton is instrumented with τ-transitions fired regularly to ensure the synchronization of time elapsing in all automata when time will be abstracted. Then, the usual region construction is performed to obtain tick automata. The proof of the equivalence of the reachability problems in both models will be finally done thanks to a technical lemma allowing to reschedule the actions of a timed automaton. Unfortunately, this approach does not apply when emptiness of channels can be tested. We discuss why it would be difficult to deal with these tests at the end of the chapter. The reciprocal reduction is intuitive (one τ is translated in one time unit) and its correctness is much simpler. It is also presented in this chapter together with the resulting undecidability result. Note that this reduction deals with emptiness tests. As a consequence, it yields a more general undecidability result than the straightforward reduction of communicating finite-state machines which already implies undecidability for non-polyforest topologies.

The chapter is structured as follows. We first explain the construction to encode timed automata by tick automata. We prove the correctness of the construction assuming that the rescheduling lemma holds. Section 11.2.2 is then devoted to the proof of this technical lemma which is nevertheless central. We draw the consequences of this reduction in Section 11.2.3, thus obtaining a decidability result. Finally, in Section 11.3 we present the reciprocal reduction which is simpler and implies undecidability results.

11.1 From continuous time to discrete time

In this section, we aim at transforming a system of communicating timed automata $\mathcal{S}$ into a system of communicating tick automata $\mathcal{S}'$ with the same topology. Of course, this transformation has to preserve reachability of final locations. Recall that the classical region construction [AD94] (recalled in Chapter 2, page 32) allows such a transformation for a single timed automaton. Moreover, it

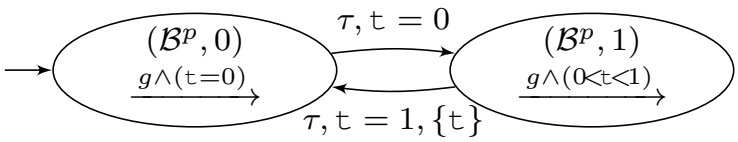


Figure 11.1: From timed to tick automata: instrumentation of a timed automaton $\mathcal{B}$ with τ-transitions.

provides a finite transition system $\mathsf{RG}(\mathcal{B})$, called the region automaton, for any given timed automaton $\mathcal{B}$ that preserves reachability of final locations. Concretely, from every path in $\mathcal{B}$ we have a path in $\mathsf{RG}(\mathcal{B})$ by removing delays. Conversely, from every path in $\mathsf{RG}(\mathcal{B})$, one can build a path in $\mathcal{B}$ by inserting adequate delays. This is however not true for a system of communicating timed automata. More precisely, there are paths in the system $\langle \mathcal{T}, M, \emptyset, (\mathsf{RG}(\mathcal{B}^p))_{p\in P}\rangle$ that are not feasible in the system $\langle \mathcal{T}, M, \mathbb{R}_+, ([\![\mathcal{B}^p]\!])_{p\in P}\rangle$. Intuitively, synchronization of processes on delays does not enable all interleavings of actions. In this section, we introduce a construction that allows to apply the region graph construction separately on each process, when the topology $\mathcal{T}$ is test-free and acyclic. Our reduction only manipulates processes locally, thus preserving the topology.

Sketch of the construction. The construction performed separately on each process can roughly be summarized as follows. We first apply a step (called "instrumentation" hereafter) that introduces ticks in order to retain enough synchronization to enable only the paths that are feasible in dense time. Then, we abstract time, replacing each timed automaton by the region automaton of its instrumented version, thus obtaining tick automata.

Formal definition of the construction.

- **Instrumentation.** Let us explain how to introduce τ-actions in each process $\mathcal{B}^p$ to perform a synchronizing tick action τ at each integer date k and at each interval $(k, k+1)$. We build an automaton $\mathsf{Instr}(\mathcal{B}^p)$, depicted in Figure 11.1 on the left, that consists in two copies (called "modes") of $\mathcal{B}^p$: $(\mathcal{B}^p, 0)$ and $(\mathcal{B}^p, 1)$. Actions occurring on integer dates k are performed in $(\mathcal{B}^p, 0)$, and those in $(k, k+1)$ happen in $(\mathcal{B}^p, 1)$. This is ensured by adding a new clock $\mathtt{t}$ and τ-transitions that switch from one mode to the other.

  Definition 11.1 (Instrumentation)**.** *The* τ-instrumentation *of the timed automaton* $\mathcal{B} = \langle L, L_I, L_F, X, \Sigma, \Delta\rangle$ *is the timed automaton* $\mathsf{Instr}(\mathcal{B}) = \langle L \times \{0,1\}, L_I \times \{0\}, L_F \times \{0,1\}, X \cup \{\mathtt{t}\}, \Sigma \cup \{\tau\}, \Delta'\rangle$*, where* $\mathtt{t} \notin X$ *and* Δ' *is defined by:*

  - $(\ell, 0) \xrightarrow{\sigma, g\wedge(\mathtt{t}=0), R} (\ell', 0) \in \Delta'$ *and* $(\ell, 1) \xrightarrow{\sigma, g\wedge(0<\mathtt{t}<1), R} (\ell', 1) \in \Delta'$ *for all rules* $\ell \xrightarrow{\sigma, g, R} \ell'$ *in* Δ*,*
  - *and* $(\ell, 0) \xrightarrow{\tau, (\mathtt{t}=0), \emptyset} (\ell, 1) \in \Delta'$ *and* $(\ell, 1) \xrightarrow{\tau, (\mathtt{t}=1), \{\mathtt{t}\}} (\ell, 0) \in \Delta'$ *for all locations* $\ell \in L$*.*

- **End of the construction.** Once the instrumentation is done, we obtain an equivalent system of tick automata by applying the region construction to each instrumented process. Formally, each process of the system of communicating timed automata is replaced by the tick automaton $\mathcal{A}^p = \mathsf{RG}(\mathsf{Instr}(\mathcal{B}^p))$, that is by the region graph of its instrumentation $\mathsf{Instr}(\mathcal{B}^p, \tau)$. The number of regions for an automaton with clocks X and maximal constant M is bounded by $|X|!.2^{|X|}.(2M+2)^{|X|}$, hence the construction is factorial in the size of $\mathcal{B}$.

The next section is devoted to the proof of the correctness of this construction with respect to the reachability problem.

11.2 Correctness of the reduction

In the previous section, we presented an abstraction of communicating timed automata by communicating tick automata, manipulating each process separately. In this section, we prove that this construction preserves the reachability of final locations, thus obtaining the following theorem.

Theorem 11.1. *Let $\mathcal{T}$ be a test-free acyclic topology. For every system of communicating timed automata $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{R}_+, (\llbracket \mathcal{B}^p \rrbracket)_{p \in P} \rangle$ with topology $\mathcal{T}$, we can produce, in exponential time, an equivalent system of communicating tick automata $\mathcal{S}' = \langle \mathcal{T}, M, \{\tau\}, (\mathcal{A}^p)_{p \in P} \rangle$ over the same topology $\mathcal{T}$, where the tick automaton $\mathcal{A}^p = \mathsf{RG}(\mathsf{Instr}(\mathcal{B}^p))$ is obtained by applying the region graph construction to $\mathsf{Instr}(\mathcal{B}^p, \tau)$.*

11.2.1 Proof of the correctness using a rescheduling lemma

Instrumentation preserves reachability Let us consider $\mathcal{S}_1 = \langle \mathcal{T}, M, \mathbb{R}_+, (\llbracket \mathsf{Instr}(\mathcal{B}^p) \rrbracket)_{p \in P} \rangle$, the system obtained from $\mathcal{S}$ by intrumenting the processes. Let $\mathtt{t}^p$ be the clock added by instrumentation in process p. From any path ϱ_1 in $\mathcal{S}_1$, we easily obtain a path ϱ in $\mathcal{S}$ by removing the τ transitions. Conversely, consider a run ϱ in $\mathcal{S}$. We need to make sure that every transition on ϱ can be taken by $\mathcal{S}_1$ despite the constraints introduced by the instrumentation on $\mathtt{t}^p$. In fact, every delay transition in ϱ can be cut into a sequence of delays and τ-transitions to ensure that for every process p, $\mathtt{t}^p = 0$ when in mode 0 and $0 < \mathtt{t}^p < 1$ when in mode 1, hence t^p does not prevent a transition to be fired. Hence, $\mathcal{S}$ has a reachable final location iff $\mathcal{S}_1$ has a reachable final location.

Synchronization over τ-transitions preserves reachability Consider system $\mathcal{S}_2 = \langle \mathcal{T}, M, \mathbb{R}_+ \cup \{\tau\}, (\mathsf{Instr}(\mathcal{B}^p))_{p \in P} \rangle$ which is similar to $\mathcal{S}_1$ except that the processes not only synchronize on dense-time delays but also on τ-transitions. Every run ϱ_2 in $\mathcal{S}_2$ is easily transformed into a run ϱ_1 in $\mathcal{S}_1$ by replacing every synchronized τ-transition by a sequence of $|P|$ τ-transitions. For the opposite direction, take any run ϱ_1 in $\mathcal{S}_1$. Since all actions performed in mode 0 occur instantaneously, τ-transitions may be interleaved with communication actions on ϱ_1. However, we can reschedule actions that occur instantaneously, taking care of the causality between communication actions. In particular, τ-transitions do not interact with communication actions. Hence, without loss of generality, we can assume that all the processes do their τ-transition sequentially when entering mode 0, then communication actions, and finally, they all do their τ-transition sequentially when leaving mode 0. Then replacing every sequence of $|P|$ τ-transitions by a synchronized τ-transition yields a path in $\mathcal{S}_2$. As a consequence, $\mathcal{S}_1$ has a reachable final location iff $\mathcal{S}_2$ has a reachable final location.

Region abstraction preserves reachability We consider the system of communicating tick automata $\mathcal{S}' = \langle \mathcal{T}, M, \{\tau\}, (\mathsf{RG}(\mathsf{Instr}(\mathcal{B}^p)))_{p \in P} \rangle$. Thanks to the region graph construction every run in $\mathcal{S}_2$ yields a run in $\mathcal{S}'$. The heart of the proof is to prove the reciprocal implication. Let us consider a run ϱ' in $\mathcal{S}'$ and prove that there is a corresponding one in $\mathcal{S}_2$.

- **Need for rescheduling.** Thanks to the region graph construction, from ϱ', we obtain a run for every process p (called local runs). The challenge to build a run of the system of communicating processes is to schedule all the actions in ϱ' on timestamps that are consistent with the guards in



Figure 11.2: Addition of τ's along a run (left) and rescheduling of a run (right).

$\mathcal{S}_2$ and that preserve dependencies between transmissions and receptions of messages. In other words, we are looking for an interleaving of the local runs which is consistent.

- **All the processes share the same mode.** Thanks to the instrumentation, all the actions between two fixed τ's in ϱ' are done in local runs either on an integer date k (mode 0 for all processes) or in an open interval $(k, k+1)$ (mode 1 for all processes).

- **Ticks partially preserve dependencies.** If a transmission of a message and its reception are separated by at least one τ-action along ϱ', then in all possible interleavings of local runs corresponding to ϱ' (necessarily respecting the synchronization over τ's), the transmission of a message is done before the reception. A problem can only occur for a transmission whose reception is done in the same interval of ticks.

  - **Mode** 0**.** If the transmission of a message and its reception are done between the same ticks and if processes are in mode 0, then they appear simultaneously (on an integer date k) in the local runs. As a consequence, it causes no problem to build the run in $\mathcal{S}_2$ from the local runs because they can freely be interleaved.
  - **Mode** 1**.** The potential conflict between guards over clocks in processes and dependencies between transmissions and receptions of messages can only appear when timed automata are in mode 1. We thus propose a way to ensure that it is always possible to schedule transmissions in channels (p, q), and then their receptions. This is depicted in Figure 11.2 on the left (before rescheduling) and on the right (after rescheduling) where the a's are emissions of p and the b's are receptions of q. The rescheduling Lemma establishes that every run of a timed automaton can be rescheduled such that integral timestamps $t_i \in \mathbb{N}$ are kept the same, and non-integral timestamps $t_i \in (k, k+1)$ belong to $k + I$ for a given $I \subseteq (0, 1)$.

    Lemma 11.1 (Rescheduling Lemma)**.** *Let $\mathcal{B}$ be a timed automaton, $\varrho = (\ell_0, v_0) \xrightarrow{d_1} (\ell_0, u_1) \xrightarrow{t_1,a_1} (\ell_1, v_1) \xrightarrow{d_2} (\ell_1, u_2) \xrightarrow{t_2,a_2} \cdots (\ell_n, v_n)$, and $I \subseteq (0, 1)$ an open interval. Then, there exists a run $\varrho' = (\ell_0, v'_0) \xrightarrow{d'_1} (\ell_0, u'_1) \xrightarrow{t'_1,a_1} (\ell_1, v'_1) \xrightarrow{d'_2} (\ell_1, u'_2) \xrightarrow{t'_2,a_2} \cdots (\ell_n, v'_n)$ such that for all $0 < i \leq n$, if $t_i \in \mathbb{N}$ then $t'_i = t_i$ and if $t_i \in (k, k+1)$ then $t' - i \in k + I$.*

  - **Please wait your turn.** Intuitively, the above lemma allows us to restrict non-integer timestamps in $(k, k+1)$ to occur in a predefined sub-interval $k + I$. Let us see how this helps in constructing a run in $\mathcal{S}_2$ from local runs in timed processes. To each process p, we associate an open interval $I_p \subseteq (0, 1)$, such that, for every channel (p, q), I_p and I_q are disjoint, and I_p comes before I_q. This is always possible on acyclic topologies. Hence, all actions of process p in $(k, k+1)$ on ϱ' can be rescheduled to occur in $k + I_p$ (according to the rescheduling Lemma), thus preserving dependencies between transmissions by p and receptions by q in intervals $(k, k+1)$.

As a consequence, local runs can be rescheduled to respect causality between transmissions and receptions. This yields a run in $\mathcal{S}_2$ where all processes synchronize on both delays and τ transitions. Hence, there is a reachable final location in $\mathcal{S}_2$ iff there is a reachable final location in $\mathcal{S}'$.

11.2.2 Proof of the rescheduling lemma

To conclude the proof of Theorem 11.1, we need to prove the correctness of the rescheduling Lemma. Intuitively, resets and guards in a timed automaton allow to enforce minimal and/or maximal delays between timestamps on a path. Since clocks are compared to integers only, it suffices to just distinguish between integral and non-integral dates. While for closed guards like $x \leq 1$ a non-integral time-point $t \in (0,1)$ would suffice to represent all non-integral dates, to accommodate open guards like $x < 1$ we need a dense interval $I \subseteq (0,1)$.

Preliminaries The proof is based on a refinement of the region equivalence for bound $M = \infty$. Two valuations v and v' are equivalent, denoted $v \sim v'$, if for all clocks x and y:

1. $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$,
2. $\{v(x)\} = 0$ iff $\{v'(x)\} = 0$, and
3. $\{v(x)\} \leq \{v(y)\}$ iff $\{v'(x)\} \leq \{v'(y)\}$.

The following Lemma is an intermediate result for the proof of the Rescheduling Lemma. It is also a good warming-up before a simple but tedious proof.

Lemma F. *For all non-negative real numbers t, t' and t'' such that $t > t'$, $t > t''$ and $0 \leq \{t'\} < \{t''\}$ we have:*

$$\{t'\} \leq \{t\} < \{t''\} \Rightarrow \{t - t'\} < \{t - t''\} \tag{11.1}$$

$$\{t\} < \{t'\} \text{ or } \{t''\} \leq \{t\} \Rightarrow \{t - t''\} < \{t - t'\} \tag{11.2}$$

Proof. First, observe that for non-negative real-numbers t and t':

$$\{t - t'\} = \begin{cases} \{t\} - \{t'\} & \text{if } \{t\} - \{t'\} \geq 0 \\ 1 + \{t\} - \{t'\} & \text{otherwise} \end{cases} \tag{11.3}$$

Let us first prove (11.1). From $\{t'\} < \{t''\}$, we have $\{t''\} < \{t'\} + 1$, hence $1 + \{t\} - \{t''\} > \{t\} - \{t'\}$. Then since $\{t'\} \leq \{t\} < \{t''\}$ it comes $\{t - t'\} < \{t - t''\}$ by (11.3).

Now, we turn to the proof of (11.2). From $\{t'\} < \{t''\}$ we deduce $\{t\} - \{t'\} > \{t\} - \{t''\}$. If $\{t''\} \leq \{t\}$, from (11.3) we obtain $\{t - t''\} = \{t\} - \{t''\} < \{t\} - \{t'\} = \{t - t'\}$. If $\{t\} < \{t'\}$, then we deduce that $1 + \{t\} - \{t'\} > 1 + \{t\} - \{t''\}$ which also leads to $\{t - t''\} < \{t - t'\}$ by (11.3).□

We are now ready to prove the rescheduling Lemma. Without loss of generality, we can assume that a run of a timed automaton $\mathcal{B}$ is an alternating sequence of delays $d_i \in \mathbb{R}_+$ and actions $a_i \notin \mathbb{R}_+$: $(\ell_0, v_0) \xrightarrow{d_1} (\ell_0, u_1) \xrightarrow{t_1, a_1} (\ell_1, v_1) \xrightarrow{d_2} (\ell_1, u_2) \xrightarrow{t_2, a_2} \cdots (\ell_n, v_n)$. We omit the timestamps on delays as they are not needed in the sequel.

Let I be an interval such that $I \subseteq (0,1)$. We show that, from every path $\varrho = (\ell_0, v_0) \xrightarrow{d_1} (\ell_0, u_1) \xrightarrow{t_1, a_1} (\ell_1, v_1) \xrightarrow{d_2} (\ell_1, u_2) \xrightarrow{t_2, a_2} \cdots (\ell_n, v_n)$, we can build a path $\varrho' = (\ell_0, v'_0) \xrightarrow{d'_1}$

$(\ell_0, u'_1) \xrightarrow{t'_1,a_1} (\ell_1, v'_1) \xrightarrow{d'_2} (\ell_1, u'_2) \xrightarrow{t'_2,a_2} \cdots (\ell_n, v'_n)$ such that $v'_0 = v_0$, and for all $i \in \{1, \ldots, n\}$, if $t_i \in \mathbb{N}$ then $t'_i = t_i$, otherwise, $t'_i \in \lfloor t_i \rfloor + I$, and $v_i \sim v'_i$ and $u_i \sim u'_i$.

Note that the integral parts of timestamps are preserved. The choice concerns only fractional parts $\{t_i\}$'s.

A proof by induction We prove by induction on the length of path ϱ that all t'_i can be chosen such that $v_i \sim v'_i$ and $u_i \sim u'_i$ for all $i \geq 0$. This is obvious for v_0 and v'_0 as they are equal. Now, we assume that $v_i \sim v'_i$ holds up to some given $i \geq 0$, and we prove that u'_{i+1} can be built such that $u_{i+1} \sim u'_{i+1}$. Observe that this entails $v_{i+1} \sim v'_{i+1}$ as v_{i+1} and v'_{i+1} are obtained from u_{i+1} and u'_{i+1} respectively by resetting the same clocks as specified by the transition labeled by the action a_{i+1}.

Notations For every clock x, let t^x denote the last timestamp before t_{i+1} when clock x has been reset. That is, t^x is the largest timestamp t_j in $\{t_0, \ldots, t_i\}$ such that x is reset on the transition labeled by a_j. In the same way, we define t'^x relatively to t'_{i+1} that we are going to build. Observe that $u_{i+1}(x) = t_{i+1} - t^x$ and $u'_{i+1}(x) = t'_{i+1} - t'^x$ for every clock x. The induction hypothesis thus implies that lemma holds for t^x and t'^x. That is: if $\{t^x\} = 0$ then $\{t'^x\} = 0$ otherwise $\{t'^x\} \in I$. Moreover, $\{u_{i+1}(x)\} = \{u_{i+1}(y)\}$ entails $\{t^x\} = \{t^y\}$ for all clocks x and y. The same holds for u'_{i+1}, t'^x and t'^y.

Condition 1 Let us prove that $\lfloor u_{i+1}(x) \rfloor = \lfloor u'_{i+1}(x) \rfloor$ for every clock x, which corresponds to condition 1 of the region equivalence. We prove that this holds for any choice of t'_{i+1} that respects the conditions of the rescheduling lemma. We have $u_{i+1}(x) = \lfloor t_{i+1} \rfloor - \lfloor t^x \rfloor + \{t_{i+1}\} - \{t^x\}$ and $u'_{i+1}(x) = \lfloor t'_{i+1} \rfloor - \lfloor t'^x \rfloor + \{t'_{i+1}\} - \{t'^x\}$. The cases where $\{t^x\} = 0$ or $\{t_{i+1}\} = 0$ are straightforward. We only detail the case where $\{t^x\} \in (0, 1)$ (which entails $\{t'^x\} \in I$ by induction) and $t_{i+1} \in (0, 1)$. We show that any choice of $\{t'_{i+1}\} \in I$ is valid. We have $-1 < \{t_{i+1}\} - \{t^x\} < 1$ hence $\lfloor t_{i+1} \rfloor - \lfloor t^x \rfloor - 1 < \lfloor u_{i+1}(x) \rfloor < \lfloor t_{i+1} \rfloor - \lfloor t^x \rfloor + 1$. In the same way, we have $-1 < \{t'_{i+1}\} - \{t'^x\} < 1$ and hence $\lfloor t'_{i+1} \rfloor - \lfloor t'^x \rfloor - 1 < \lfloor u'_{i+1}(x) \rfloor < \lfloor t'_{i+1} \rfloor - \lfloor t'^x \rfloor + 1$. Now, since $\lfloor t_{i+1} \rfloor = \lfloor t'_{i+1} \rfloor$, $\lfloor t^x \rfloor = \lfloor t'^x \rfloor$, and 0 is the only integer strictly between -1 and 1, we deduce that $\lfloor u_{i+1}(x) \rfloor = \lfloor u'_{i+1}(x) \rfloor$.

Condition 2 **and** 3 We now prove that conditions 2 and 3 of the region equivalence hold. Let $X_0, \ldots, X_k \subseteq X$ define a partition of the clocks according to their fractional part in the valuation v_i.

- **Partitioning** For each $x, y \in X_j$, $\{v_i(x)\} = \{v_i(y)\}$, for each $x \in X_j$ and $y \in X_{j-1}$, $\{v_i(y)\} < \{v_i(x)\}$, and $\bigcup_{j=0}^{k} X_j = X$. Observe that v_i and v'_i define the same partition of clocks as $v_i \sim v'_i$. This partition is depicted in Figure 11.3 to the left. As time elapses from v_i and v'_i, the fractional part of clock valuations increases and the ordering of partitions changes in a circular way. Some clocks, say $X_0, \ldots, X_{j-1}$ have their fractional part increased, whereas some others, say $X_j \ldots, X_k$ have their fractional part decreased as they have been set back to 0 meanwhile. Assume that the ordering of fractional part of the clocks in u_{i+1} is as depicted in Figure 11.3 to the right.

- **Rephrasing** We now show that $\{t'_{i+1}\}$ can always be chosen in such a way that u'_{i+1} has the same ordering of the fractional part of the clocks as u_{i+1}, which will conclude the proof that $u_{i+1} \sim u'_{i+1}$.

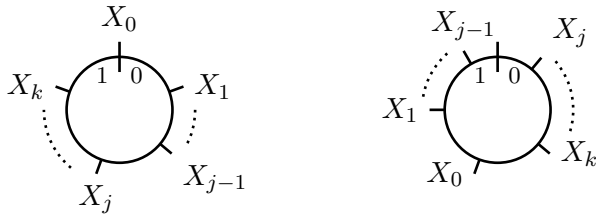


Figure 11.3: The ring of fractional parts before (left) and after (right) time elapses.

– **Trivial partition case** We first consider the case when the partition only contains the single set X. As all the clocks have the same fractional part, only condition 2 of the region equivalence needs to be considered.
  * If $\{u_{i+1}(x)\} = 0$ for every clock x, we simply choose $\{t'_{i+1}\} = \{t'^x\}$ which yields $\{u'_{i+1}(x)\} = 0$. By induction, $\{t'_{i+1}\}$ satisfies the lemma.
  * If $\{u_{i+1}(x)\} > 0$ for every clock x, choosing $\{t'_{i+1}\} \neq \{t'^x\}$ guarantees that $\{u'_{i+1}(x)\} > 0$ too. We need to show that there always exists such a solution. From $\{u_{i+1}(x)\} > 0$, we obtain $\{t_{i+1} - t^x\} > 0$, hence we cannot have $\{t_{i+1}\} = 0$ and $\{t^x\} = 0$ at the same time. If $\{t_{i+1}\} = 0$ then $\{t^x\} > 0$, hence $\{t'_{i+1}\} = 0$ is a solution since $\{t'^x\} > 0$ by induction hypothesis. Conversely, if $\{t_{i+1}\} \in (0, 1)$, then we can choose any $\{t'_{i+1}\} \in I$ distinct from $\{t'^x\}$ (recall $\{t'^x\} = \{t'^y\}$ for all clocks x and y).

– **General case** Now, we consider a partition $X_0, \ldots, X_k$ of the clocks in v_i and v'_i, with $k \geq 1$, and the partition $X_j, \ldots, X_k, X_0, \ldots, X_{j-1}$ in u_{i+1} as depicted in Figure 11.3. Let us first focus on the case when $\{u_{i+1}(x)\} = 0$ for $x \in X_j$. As $u'_{i+1}(x) = t'_{i+1} - t'^x$, for $\{u'_{i+1}(x)\} = 0$ it must be the case that $\{t'_{i+1}\} = \{t'^x\}$. By induction hypothesis, this value of $\{t'_{i+1}\}$ satisfies the lemma.

  Now consider the case where $\{u_{i+1}(x)\} > 0$ for $x \in X_j$. As illustrated on Figure 11.3 to the right, we need to make sure that, in valuation u'_{i+1}, the clocks in X_j have the smallest fractional part and the clocks in X_{j-1} have the largest one. This is ensured by condition $\{u'_{i+1}(x)\} < \{u'_{i+1}(y)\}$ for $x \in X_j$ and $y \in X_{j-1}$, which translate as:

$$\{t'_{i+1} - t'^x\} > 0 \quad \text{and} \quad \{t'_{i+1} - t'^x\} < \{t'_{i+1} - t'^y\}. \tag{11.4}$$

  We distinguish two cases depending on whether $\{t'^y\} > \{t'^x\}$ or $\{t'^y\} < \{t'^x\}$. Let us consider the first case. From Lemma F and inequalities (11.4), we need to find a value of $\{t'_{i+1}\}$ such that $\{t'^x\} < \{t'_{i+1}\} < \{t'^y\}$. By induction hypothesis we have $\{t'^y\} \in I$ and the following two cases for $\{t'^x\}$:
  * either $\{t'^x\} = 0$, then $\{t^x\} = 0$ by induction, hence $\{t_{i+1}\} > 0$ as $\{u_{i+1}(x)\} > 0$. Since $\{t_{i+1}\} \in (0; 1)$ we must choose $\{t'_{i+1}\}$ in $I = (a, b)$. Taking $\{t'_{i+1}\} = \frac{a+\{t'^y\}}{2}$ fulfills all the requirements.
  * or $\{t'^x\} \in I$. Then choosing $\{t'_{i+1}\} = \frac{\{t'^x\}+\{t'^y\}}{2}$ yields a solution.

  It remains to consider the case when $\{t'^y\} < \{t'^x\}$. Applying Lemma F on (11.4) yields two sets of solutions: $\{t'_{i+1}\} < \{t'^y\}$ or $\{t'^x\} \leq \{t'_{i+1}\}$.
  * If $\{t_{i+1}\} \in (0; 1)$, then $\{t'_{i+1}\} = \frac{\{t'^x\}+b}{2}$ is a solution as $\{t'^x\} \leq \{t'_{i+1}\}$ and, by induction hypothesis, $\{t'^x\} \in I$ since $\{t'^y\} < \{t'^x\}$ (i.e. $\{t'^x\} \neq 0$).

- Now, if $\{t_{i+1}\} = 0$ we have $\{t^y\} > 0$. Indeed, as $y \in X_{j-1}$, we have $\{u_{i+1}(y)\} = \{t_{i+1} - t^y\} > 0$ and $\{t^y\} = 0$ entails $\{t_{i+1}\} > 0$, a contradiction. By induction hypothesis, from $\{t^y\} > 0$ we get $\{t'^y\} > 0$. Hence, we can pick $\{t'_{i+1}\} = 0$ which satisfies $\{t'_{i+1}\} < \{t'^y\}$.

Finally, it remains the case when the ordering of fractional parts is the same in v_i and u_{i+1}. Then, considering $X_j = X_0$, and $X_{j-1} = X_k$ yields a solution for $\{t'_{i+1}\}$ as stated above.

11.2.3 Consequences

We reduced the reachability problem in communicating timed automata with a test-free and acyclic topology to the reachability problem in communicating tick automata with the same topology. As a consequence of this reduction together with Theorem 10.3 and Corollary 10.1, we obtain the following theorem.

Theorem 11.2 (Characterization of decidable test-free topologies)**.** *Given a test-free topology $\mathcal{T}$, the reachability problem for systems of communicating timed automata with topology $\mathcal{T}$ is decidable if and only if $\mathcal{T}$ is a polyforest.*

Moreover, the reachability problem for systems of communicating timed automata with test-free polyforest topology is EXSPACE-hard and in 2EXPSPACE.

While the reachability problem is known to be decidable for a system of two communicating timed automata with only one channel and emptiness test [KY06], that proof does not preserve the topology and it looks hardly adaptable to arbitrary polyforest topologies. Unfortunately, the rescheduling approach does not work when emptiness of channels can be tested because a rescheduling can change the result of an emptiness test. Therefore, the above decidability result is not as general as Theorem 10.3 for tick automata. We discuss this limitation in Section 11.4.

11.3 Reciprocal reduction and its consequences

In this section, we translate the undecidability cases of Theorem 10.3 to communicating timed automata. To do so, we make the reduction from tick to timed automata in a very intuitive way.

Reduction Given a system of communicating tick automata $\mathcal{S}$, we produce an equivalent system of communicating timed automata $\mathcal{S}'$, over the same topology. The synchronization on τ's is easily simulated using clocks in $\mathcal{S}'$ by ensuring that all the processes elapse 1 time unit exactly when they (synchronously) perform a τ in $\mathcal{S}$. Thus, every run in $\mathcal{S}$ has a corresponding run in $\mathcal{S}'$. For the converse to hold, we have to make sure that for every run of $\mathcal{S}'$, all the processes perform the same number of τ's on the corresponding run of $\mathcal{S}$. In fact, it is always possible to make uniform the number of ticks in processes. Indeed, this is ensured by condition (9.1) (in page 176) since we require that timed processes can always delay from their final locations, while staying in final locations.

The following theorem thus follows from Theorem 10.3.

Theorem 11.3 (Undecidability)**.** *Given a topology $\mathcal{T}$ with two testable channels in the same weakly-connected component, the reachability problem for systems of communicating timed automata with topology $\mathcal{T}$ is undecidable.*

The simple topology $p \to q \to r$ was known to be undecidable when both channels can be tested for emptiness [KY06]. Theorem 11.3 strongly generalizes this result establishing undecidability for every topology containing at least two testable channels in the same weakly-connected component.

11.4 Abstraction of communicating timed automata with emptiness tests is difficult

In this section we discuss why our abstraction (presented in Section 11.1) does not work with channels with emptiness tests and why it seems difficult to find a suitable abstraction that preserves the topology. Notice that an abstraction that does not preserve the topology is known for the particular case of a channel with distinct sender and receiver [KY06].

11.4.1 Our construction is not sound for emptiness test

We propose the simple example in Figure 11.4. From top to bottom, there are a sender and a receiver, communicating via a channel c. We can easily verify that there is no global run in this system. Indeed, due to timing constraints, the actions along a global run have to be in the following order: $c!a; c?a; c == \varepsilon; c!a; c == \varepsilon; c?a$. Then the second emptiness test fails because c is not empty. Hence the receiver cannot reach its final location. On the contrary, the system of communicating tick au-

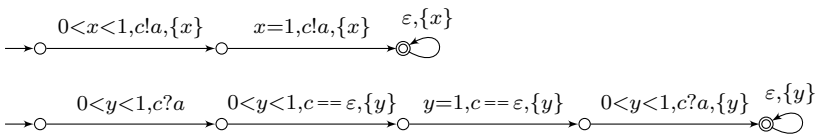


Figure 11.4: A counter-example to our abstraction with emptiness test.

tomata obtained by applying the construction in Section 11.1 has a global run that reaches the final locations. This system is depicted in Figure 11.5. The global run corresponds to the sequence of actions $c!a; c?a; c == \varepsilon; \tau; c == \varepsilon; c!a; c?a$ where both processes synchronize on τ. Observe that this global run cannot be re-scheduled in the spirit of the Rescheduling Lemma. Indeed, both real-time constraints and dependencies between the communication actions prevent to swap actions $c == \varepsilon; c!a$ into $c!a; c == \varepsilon$.

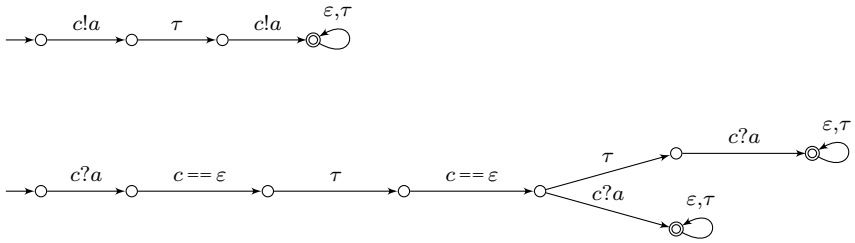


Figure 11.5: A counter-example to our abstraction with emptiness test.

In fact, even with a single channel, scheduling all the emissions of a one time unit before all the receptions, emptiness tests can becomes false. We then would like to schedule differently in order to preserve the successful emptiness tests. For instance, if the sender emit a and b and the receiver can receive a but is able to receive b only after an emptiness test and an internal action, we would like to schedule the reception of a and the emptiness test before the emission of b. A first idea would thus be to try to allocate several slots to each process.

11.4.2 Why soundness is hard to achieve

Our abstraction is based on the possibility to allocate one slot per time unit (the interval I in the rescheduling Lemma) to each process in the system. In the previous section, we have seen that in presence of emptiness test, one slot per process may not be sufficient. We now show that we cannot even find a bound on the number of slots per time unit needed by each process.

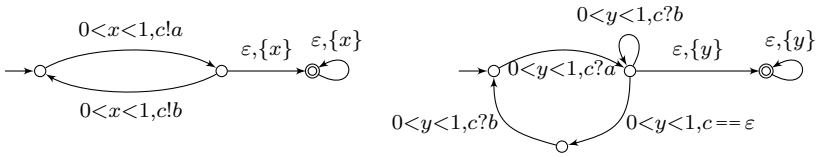


Figure 11.6: A counter-example to our abstraction with emptiness test.

Figure 11.6 shows an example with a sender p (left) and a receiver q (right) that communicate via a channel $c = (p, q)$. Consider a global run of the system where the sender p performs actions $c!a; c!b$ while the receiver q does actions $c?a; c == \varepsilon; c?b$. Obviously, q has to perform the emptiness test $c == \varepsilon$ between the two emissions by p. Observe that both processes can iterate this behavior. Finally, all these actions occur in one time unit. This shows that the number of slots needed by p and q depends on the number of iterations on their respective loops. Thus there may not be an uniform choice of slots in presence of emptiness tests.

Notice that this is due to a convergence phenomenon but not necessary to Zeno behaviors. Adding loops that reset the clocks on the initial locations of both process, we could let one time unit elapse infinitely often, but the problem would remain the same.

Conclusion

In this chapter, we translated the characterization of the topologies of systems of communicating processes for which reachability is decidable from communicating tick automata to dense timed processes. To do so, we introduced a rescheduling lemma allowing, in timed automata whose accepting locations are reachable, to build runs in which actions are done either on integral timestamps or on timestamps whose fractional part is in any fixed interval. We also generalized the undecidability result of [KY06], doing a reduction from discrete time to dense time. Finally, we illustrated the limitations of the rescheduling approach with respect to the tests of emptiness of channels (which corresponds to the urgency in [KY06]). We thus exhibited a simple example constituted of one channel with a sender and a receiver illustrating the impossibility to bound the number of interleavings of sequences of their respective actions. Note that this example is based on convergence phenomena that we studied in the previous part in timed automata with frequencies. It could be interesting to investigate some restrictions using the notion of forgetfulness, which may permit the rescheduling lemma or a variant to apply.

Conclusion

We have studied the decidability and complexity of the reachability problem for communicating timed processes synchronizing over discrete or continuous time. In discrete time, we gave a complete characterization of decidable topologies with emptiness tests, as well as a tight connection with Petri nets in the test-free case which allows to conclude that for polyforest topologies, the reachability problem is EXSPACE-complete. Our approach for the characterization can be seen as a generalization of the proof in [KY06] where dense time was only used in a discrete way. In dense time, we proved the decidability for polyforest test-free topologies which is the same result as for communicating finite-state machines. Moreover, we generalized the undecidability result of [KY06] to arbitrary weakly-connected topologies containing two testable channels. The rescheduling approach does not work in the presence of emptiness tests, to lift the characterization of decidable topologies from discrete time to dense time. Nevertheless, an interesting open problem is to get an equivalent system of tick automata by another approach.

Part VI

Conclusion and Future Works

Conclusion and Future Work

Determinization of timed automata

Contribution To face the unfeasibility of determinization of timed automata in general, we proposed a game approach that, given a timed automaton and resources (number of clocks and maximal constant), allows one to find a good reset policy for the determinization of the timed automaton with these resources. Every winning strategy yields a deterministic equivalent and losing strategies produce a deterministic approximation with good properties (over approximation or abstraction). The approach improves the two existing methods [BBBB09, KT09]: it exactly determinizes strictly more timed automata, and approximates more precisely in case the determinization is not exact.

A prototype has been implemented during a visit at Aalborg University in the team of Kim G. Larsen and in particular thanks to the help of Peter Bulychev. For the moment, the implementation is a bit rough. It implements the one-the-fly algorithm from [LS98] to avoid to build all the game if possible.

The unfeasibility of determinization of timed automata can be annoying in the verification process. Our game-approach yields a pragmatic solution to this fundamental problem, allowing to deal with the impossibility to determinize all timed automata by performing an approximation when needed. Thanks to the very expressive relations between clocks, our approximate determinization is built in a clever way.

Implementation and heuristics The theoretical problem is hard and it would be of interest to develop a more efficient tool which could be used or integrated to existing tools. We have several ideas to develop.

- **Greedy strategy for determinizable classes** Our game-based algorithm allows to determinize all timed automata of known determinizable classes. For these timed automata, the strategy is simple and the deterministic timed automaton can be built in a greedy way. Roughly, each time there is a clock of X which is reset (in the non-deterministic timed automaton), we reset one clock of Y (in the deterministic timed automaton) preserving the fact that each clock of X is exactly expressed (equal up to an integer). To do so, we choose to reset the first clock of Y which is no longer used to express any clock of X, or which is used to express clocks whose values are larger than the maximal constant, or which is used to express clocks having integral values (hence it can be reset without losing the expression of these clocks up to an integer). Such a strategy is winning for timed automata of classes known to be determinizable. A first step in order to make the implementation usable would be to first try this strategy. Indeed, most of the realistic real time systems can be modeled by timed automata of these classes.

- **Beyond the greedy strategy** Even if a lot of non-deterministic timed automata are determinizable with our greedy approach, we need to improve the performances of the general algorithm.

Indeed, on the one hand, it is nice to be able to always produce a deterministic timed automaton even if it is an approximation; on the other hand, the resources imposed for the determinization can be restricted in such a way that the greedy strategy cannot be used. In this case, we search a good tradeoff between the precision of the game approach and the computation time. Starting with the greedy strategy until a deadlock is reached, can be a first option. Then, we could try several heuristics to obtain greedy strategies hoping to find winning strategies or "good" losing strategies. For example, one can minimize the imprecision of the expression of the clocks of X by the clocks of Y in different ways. One can minimize the number of clocks of X which are no longer expressed up to an integer by clocks of Y. One can also consider the size of the imprecision. One thus prefers the relation $1 < x - y < 2$ to the relation $0 < x - y < 3$. It is then possible to minimize the imprecisions in different ways, considering the maximal imprecision for a clock of X, or the sum of the imprecisions, or the average. We do not know if one of these heuristics is better than the other ones. The comparison of losing strategies needs also to be investigated. Sometimes, the language inclusion of two deterministic approximations can be decided, but they can be incomparable. Depending on the context, we would like to develop adequate criteria, and optimize heuristics with respect to them.

- **Finding small strategies** Beyond the size of the game or of the part of the game that we build, the size of the resulting deterministic timed automata is also very important. We would like to design heuristics to optimize this parameter. For example, we could simply add, in the resources, a bound over the number of locations for the deterministic timed automaton which is built. In this case, it would be possible to produce no result. It could of course be a first try before using another version of the approach. Another idea would be to define simulation relations between states in the game allowing to use the same strategy from the state which is simulated as from the other. We thus hope that the deterministic timed automaton could be quotiented into a smaller one.

Extensions to other models Extensions of timed automata have been investigated, such as timed automata with data [dLAMJM11], timed automata with costs [ATP01, BFH$^+$01] or even linear hybrid systems [ACHH92]. We would like to adapt our game approach to deal with these models. The states of the game would need to be refined, but studying the approximation mechanism and some extensions of the relations (in particular for linear hybrid systems) could be of interest.

Off-line test selection for non-deterministic timed automata using test purposes

Contribution We provided a general formal framework for the generation of test cases from specifications given as non-deterministic timed automata. In this approach, we use test purposes expressed thanks to timed automata able to observe clocks of the specification. Problems due to non-determinism of the specification are fixed thanks to our game approach for the determinization. It ensures that even if the resulting deterministic specification is an io-abstraction, the conformance relation is preserved. More precisely, sound test cases produced from the computed deterministic specification are always sound for the non-deterministic specification. The risk due to the approximation is to miss some non-conformances, that is to emit false positive verdicts.

Our approach is very general and provides at once, a formal setting and a procedure for the off-line generation of test cases. To our knowledge, this is the most complete framework for this concrete problem.

Control of the reachability of test purposes During the execution of a test case, the tester can control inputs of the implementation, but outputs are not controllable. As a consequence, some test purposes cannot be satisfied even if the implementation conforms to a specification which can satisfy the purpose. In [DLLN09], a game approach is proposed to select controllable executions of the test cases. The risk is then to miss some or even all traces. Our approach allows one to lose the game and produce an Inconc verdict when this happens. Nevertheless, we can optimize the chances to satisfy the test purpose. In our framework, we prune the controllable actions of the test cases leading to states which are not co-reachable from the purpose, but we could improve the method. For example, when two inputs are possible, one input can lead in a controllable path whereas the other does not. We then could perform a preprocessing in order to compute states from which the reachability of the test purpose is controllable, as well as the paths to execute.

Coverage criterion to lead generation of test purposes Test purposes allow one to select some behaviors to test. In our framework, we assume that test purposes are given. The generation of a suitable set of test purposes is a problem in itself. A syntactic coverage criterion (*e.g.* edge coverage, and see [UL10] for more examples) intrinsically defines a set of test purposes. Naturally, depending on the context, expected properties can be different, but we would like to define coverage criteria to guide the generation of test purposes. Then, there will be a tradeoff between the precision of the coverage criterion and the controllability of the reachability of the purposes. It would be of particular interest to discuss with industrial partners in order to understand their practices and concrete needs.

Extension to networks of timed automata More and more systems are distributed. They can be modeled by communicating processes for example by synchronized exchanges of messages or via channels. We saw, in Part V, that their verification is very difficult. It is then interesting to generate test cases for such a model, even if the scalability remains the main problem. Specifications can be defined separately for each process. Of course, the global system can be tested as a single transition system, but the entire model could have a huge size or be infinite. The goal of the approach would be to distribute the generation of test cases and thus avoid to compute or to widely explore the product of all the processes. Even without timing aspects, this problem is hard. First of all, the usual conformance relation is not compatible with the composition, that is two implementations can conform to their respective specifications whereas the product of them does not conform to the product of the specifications, because of some deadlocks. Then, a possible outline for the approach would be to generate test cases separately for each process and to use them for the generation of test cases for the global system. We would like to investigate possibilities to adapt our approach to such settings, even if we know that it is a long term objective.

Frequencies in timed automata

Contribution We defined new quantitative semantics for the acceptance of infinite timed words in timed automata. The frequency of a run is the proportion of time elapsed in accepting location along it. A run is thus said accepting if it satisfies a fixed frequency-based constraint. A word is accepted if there is at least one accepting run reading it. With the motivation to decide language problems such as emptiness or universality, we studied the set of frequencies in timed automata. We first presented techniques applying only to one-clock timed automata to compute the bounds of the set of frequencies, and then extended our results to timed automata with several clocks, assuming that there were no convergence phenomena. The computation of the bounds of the set of frequencies in a timed automaton permits to decide emptiness, and even universality if the timed automaton is

deterministic. Finally, we reduced the universality problem for timed automata with Büchi semantics to the universality problem with frequencies. The latter problem is thus non-primitive recursive for one-clock timed automata and undecidable with several clocks. We also proved the decidability of the universality for Zeno words in one-clock timed automata with positive frequencies, but the non-Zeno case remains open.

Modeling quantitative aspects in timed automata is primordial to answer to questions about energy consumptions or failure rates for example. This work focuses on a particular case of double-priced timed automata and for their study, develops novel techniques. These techniques permit to investigate deeply the link between runs in a timed automaton and runs in its corner-point abstraction. They are generic and could be adapted to other contexts.

Remaining open questions Beyond the decidability of the universality for non-Zeno words in one-clock timed automata with frequencies, several questions remain open, in which we are interested but for which we do not have concrete ideas. First, we proved the decidability of the universality for Zeno words in one-clock timed automata with positive frequencies, but the proof does not trivially extend to other frequency-based constraints. Moreover, we showed the undecidability of the universality problem for timed automata with frequencies, but we do not know whether the universality for forgetful and/or strongly non-Zeno timed automata with frequencies is undecidable. Generally, we are still looking for restrictions allowing to decide universality with frequencies. On the other hand, concerning the decidability of the emptiness problem, we would like to relax non-convergence assumptions. A first step would be to deal with Zeno behaviors, that is to relax the strong non-zenoness assumption. For one-clock timed automata, we succeeded in dealing with them thanks to careful inspections of cases. We would be interested in investigating whether the techniques can be extended to timed automata with several clocks. Finally, relaxing the forgetfulness assumption on timed automata would be of interest, we hope that it could help to better understand the convergence phenomena.

Determinization preserving some properties of the frequencies The determinization of timed automata is not possible in general, we discussed this fact in Part III of the document. Nevertheless, some classes are known to be determinizable. Note that this determinizability is valid only for the usual semantics for finite words. Indeed, even without time, deterministic Büchi automata are strictly less expressive than non-deterministic ones. Similarly to Büchi automata, the determinization of timed automata with frequencies is impossible in general. We would like to find some restriction allowing the determinization not necessarily preserving exact frequencies, but possibly preserving other properties such as "larger than a threshold" to preserve a frequency-based language. Another important operation which we would like to be able to perform, would be the complement. The main interest of the complement would be to decide the universality of the language, thus reducing it to emptiness of the complement.

Extension to several frequencies A natural extension of our frequency-based semantics is to consider frequencies of several subsets of locations. Acceptance conditions could thus be of the form $\mathsf{freq}_1 \geq \lambda_1 \wedge \mathsf{freq}_2 \geq \lambda_2$. The emptiness of the languages defined in this way would then be harder to decide (if decidable at all). A first step could be to investigate such a model without time where transitions have all the same weight. This is quite close to [CDHR10] where generalization of mean-payoff conditions are studied in games.For one-clock timed automata, we contracted time in accepting locations and dilated time in non-accepting location to compute lower bounds of the sets of frequencies. With two frequencies, this is not as simple, but there may exist variants allowing to decide the

existence of a run satisfying at once two constraints over two different frequencies.

Forgetfulness of timed automata

Contribution The notion of forgetfulness has been introduced in [BA11] to detect cycles of timed automata allowing to read timed languages with positive entropy, that is cycles along which there is no forced convergence. Observing convergences asks for more and more precise clocks, thus these behaviors are not realistic in an implementability point of view. We assumed the forgetfulness of cycles to compute the set of frequencies of a large class of timed automata. Then, we established technical lemmas allowing to make a tight link between runs of a timed automaton and runs of its corner-point abstraction. In this way, given a run of the abstraction, one can build a very similar run in the timed automaton, preserving the frequency value.

Forgetfulness is a fundamental notion close to non-zenoness, allowing to detect convergence phenomena. It has been initially used to characterize timed automata whose languages have a positive entropy [BA11]. Since the publication of our work, it has also been useful for robust control [SBMR13]. Moreover, convergences seem to be behind the limitations of other results such as for the model checking of stochastic timed automata [BBBM08].

Open questions about forgetful timed automata Forgetful timed automata have been recently introduced and several questions have to be studied. For example, this class is clearly closed under union, but is it closed under other operations? Can we decide the universality of timed languages for a given semantics? Does this class have nice properties useful in other contexts? In summary, we are very impatient to broaden our knowledge about forgetful timed automata.

Using forgetfulness for some other problems Forgetfulness already was useful in another context: it has been used to characterize timed automata which are robustly controllable [SBMR13]. In other words, the goal is to distinguish timed automata where a Büchi condition can be satisfied in a robust way; that is even when the chosen delays are systematically perturbed by an adversary by a bounded parametrized amount. Timed automata for which there exists such a positive parameter are proved to be exactly timed automata containing an accepting lasso whose cycle is forgetful.

On the other hand, we currently work on the model-checking of a robust variant of LTL, considering that only paths whose languages have a positive entropy matter. The formula is thus satisfied if and only if there is no lasso counterexample with a forgetful cycle. To search such a counter-example, we then prove that forgetfulness of a cycle implies its discretizability, and encode the search in a SAT instance. The problem can thus be solved using a SAT-solver.

Finally, the quantitative model checking of timed automata with a probabilistic semantics is restricted to one-clock timed automata, for the moment, but this limitation seems to be due to convergence phenomena [BBBM08]. It would be of interest to investigate the possibility to extend the approach to forgetful timed automata. Generally, we are still looking for other contexts in which forgetfulness could be relevant and helpful.

Communicating timed processes

Contribution The most notable result is the characterization of topologies for which the reachability problem is decidable in communicating timed automata. As for communicating finite-state machines, the reachability problem is decidable if and only if the topology is a polyforest. This result can seem to contradict the main theorem of [KY06] which establishes the undecidability for pipelines if and

only if there are at least three processes. In fact, models are slightly different. Indeed, in [KY06], the undecidability result relies on the combination of the synchronization over time, and of the urgency of receptions from channels, which disables internal actions when receptions are possible. Considering urgent and non-urgent channels, we extended the result of [KY06]. On the one side, for communicating discrete timed processes, we completely characterized the decidable topologies. On the other side, we partially translated this characterization to dense time, proving that polyforests, without urgency, are decidable thanks to a rescheduling lemma. Unfortunately, this technical lemma does not deal with urgency. As a consequence, the decidability of polyforest topologies with some urgent channels remains an open question. Nevertheless, we also extended the undecidability result of [KY06], showing that topologies which are not polyforests, or with a weakly connected component with at least two urgent channels, are undecidable.

The remaining open question Our reduction from discrete time to dense time does not deal with urgency. The decidability of polyforests with at most one urgent channel per weakly connected component is still open. We discussed the limitations of the rescheduling approach, observing some convergence phenomena. Such convergences have been also observed in timed automata with frequencies, hindering the computation of the set of frequencies in timed automata. In a single timed automaton, convergence phenomena can be detected thanks to the notion of forgetfulness. In order to complete the characterization of decidable topologies when urgent channels are allowed, two ideas are thus possible. Either we choose to study these convergences, to isolate them, and try an approach similar to the one we had for frequencies, or we investigate other approaches to perform a different reduction from discrete time to dense time, maybe not preserving the topology. Indeed, our reduction is distributed in the sense that it manipulates each process separately. We conjecture that a reduction in the general case is possible, hence we currently investigate this question.

Other ideas to gain decidability of the reachability problem In this document, we extended the characterization of decidable topologies from communicating finite-state machines to communicating timed processes. For communicating finite-state machines, the restriction to decidable topologies (*i.e.* polyforest topologies) is strong. Indeed, in polyforest communicating finite-state machines, it is sufficient to consider one-bounded channels to decide the reachability problem. This is not the case when processes are synchronized by time elapsing. Several other restrictions over the communicating finite-state machines have been studied to obtain the decidability of the reachability problem. It could be of interest to study other decidable classes of communicating finite-state machines in order to extend some results to communicating timed automata. For example, when we consider bounded channels, these can be seen as a finite memories which do not increase the complexity of models. The decidability is also straightforward for communicating timed processes with bounded channels. Nevertheless, the synchronization could lead to new decidable classes inspired by other works such as the language restriction for channels in [JJ93], which, roughly, forbids embedded loops in channels languages.

Extension to systems with clock drifts Distributed timed automata with independently evolving clocks have been introduced in [ABG$^+$08]. In this model timed automata exchange information only observing clocks of other processes. One of the main results considers a universal semantics over this model: an untime word is accepted if for all drifts of clocks, there is an accepted run reading this word. The existence of such a word is then proved to be an undecidable problem. We are very interested in such models with clock drifts. We are currently investigating the model of communicating timed

processes where time rates are local: the synchronization is loose. The universal semantics imposes a much stronger condition for runs to be accepted, it seems to help to decide whether there exists an accepted run. In particular, we can decide, thanks to a non-trivial construction, whether a fixed run can be executed whatever the drifts. To decide whether there exists an accepted word for the universal semantics, the problem is different, since we should consider several runs for a single word. In the long term, we would like to investigate other semantics, and particularly with boundedness assumptions over the drifts for communicating timed processes via channels, as it is done in [ABG$^+$08] with observations of clocks.

Bibliography

[AAC12] Parosh A. Abdulla, Mohamed F. Atig, and Jonathan Cederberg. Timed lossy channel systems. In *Proceedings of the 32th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'12)*, volume 18 of *LIPIcs*, pages 374–386. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[ABG07] S. Akshay, Benedikt Bollig, and Paul Gastin. Automata and logics for timed message sequence charts. In *Proceedings of the 27th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2007.

[ABG+08] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2008.

[ACHH92] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1992.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *Proceedings of the 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 L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AD09a] Eugene Asarin and Aldric Degorre. Volume and entropy of regular timed languages: Analytic approach. In *Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2009.

[AD09b] Eugene Asarin and Aldric Degorre. Volume and entropy of regular timed languages: Discretization approach. In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR'09)*, volume 5710 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2009.

[AD10] Eugene Asarin and Aldric Degorre. Two size measures for timed languages. In *Proceedings of the 30th IARCS Annual Conference on Foundations of Software Tech-*

nology and Theoretical Computer Science (FSTTCS'10), volume 8 of *LIPIcs*, pages 376–387. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[ADOW05] Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, and James Worrell. Decidability and complexity results for timed automata via channel machines. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005.

[ADR+11] Parosh A. Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *Proceedings of the 9th International Colloquium on Formal Modeling and Analysis of Timed Systems (FORMATS'11)*, volume 6919 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2011.

[AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[AHKV98] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR '98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proceedings of the 5th IFAC Symposium on System Structure and Control (SSSC'98)*, pages 469–474. Elsevier Science, 1998.

[ATP01] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001.

[BA11] Nicolas Basset and Eugene Asarin. Thin and thick timed regular languages. In *Proceedings of the 9th International Colloquium on Formal Modeling and Analysis of Timed Systems (FORMATS'11)*, volume 6919 of *Lecture Notes in Computer Science*, pages 113–128. Springer, 2011.

[BB05] Laura Brandán Briones and Ed Brinksma. A test generation framework for quiescent real-time systems. In *Proceedings of the 4th International Workshop on Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2005.

[BBB+08] Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Marcus Größer. Almost-sure model checking of infinite paths in one-clock timed automata. In *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS'08)*, pages 217–226. IEEE, 2008.

[BBBB09] Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye. When are timed automata determinizable? In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09)*, volume 5556 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2009.

[BBBM08] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Quantitative model-checking of one-clock timed automata under probabilistic semantics. In *Proceedings of the 5th International Conference on the Quantitative Evaluation of Systems (QEST'08)*, pages 55–64. IEEE, 2008.

[BBBS11] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Amélie Stainer. Emptiness and universality problems in timed automata with positive frequency. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP'11)*, volume 6756 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2011.

[BBBS13] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Amélie Stainer. Emptiness and universality problems in timed automata with positive frequency. CoRR arXiv:1309.2842, 2013.

[BBL08] Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):3–23, 2008.

[BCD05] Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2005.

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

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

[BDL+12] Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Axel Legay, Guangyuan Li, Danny Bøgsted Poulsen, and Amélie Stainer. Monitor-based statistical model checking for weighted metric temporal logic. In *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'12)*, volume 7180 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2012.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[BGP96] Béatrice Bérard, Paul Gastin, and Antoine Petit. On the power of non-observable actions in timed automata. In *Proceedings of the 13th Annual Symposium on Theoretical*

Aspects of Computer Science (STACS'96), volume 1046 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 1996.

[BJSK11] Nathalie Bertrand, Thierry Jéron, Amélie Stainer, and Moez Krichen. Off-line test selection with test purposes for non-deterministic timed automata. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, volume 6605 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2011.

[BJSK12] Nathalie Bertrand, Thierry Jéron, Amélie Stainer, and Moez Krichen. Off-line test selection with test purposes for non-deterministic timed automata. *Logical Methods in Computer Science*, 8(4:8), 2012.

[Bon11] Rémi Bonnet. The reachability problem for vector addition system with one zero-test. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *Lecture Notes in Computer Science*, pages 145–157. Springer, 2011.

[Bou09] Patricia Bouyer. *From Qualitative to Quantitative Analysis of Timed Systems*. Mémoire d'habilitation, Université Paris 7, Paris, France, January 2009.

[BSJK11a] Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen. A game approach to determinize timed automata. In *Proceedings of the 14th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'11)*, volume 6604 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2011.

[BSJK11b] Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen. A game approach to determinize timed automata. Research Report 7381, INRIA, Rennes, France, July 2011.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CCH+05] Arindam Chakrabarti, Krishnendu Chatterjee, Thomas A. Henzinger, Orna Kupferman, and Rupak Majumdar. Verifying quantitative properties using bound functions. In *Proceedings of the Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*, volume 3725 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2005.

[CDE+10] Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In *Proceedings of the 21th International Conference on Concurrency Theory (CONCUR'10)*, volume 6269 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2010.

[CDH08] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. In *Proceedings of the 22nd International Workshop on Computer Science Logic (CSL'08)*, volume 5213 of *Lecture Notes in Computer Science*, pages 385–400. Springer, 2008.

[CDHR10] Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In *Proceedings of the 30th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *LIPIcs*, pages 505–516, 2010.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CF05] Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190, 2005.

[CFI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

[CHR02] Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In *Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control (HSCC'02)*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2002.

[CHSS12] Lorenzo Clemente, Frédéric Herbreteau, Amélie Stainer, and Grégoire Sutre. Reachability of communicating timed processes. CoRR arXiv:1209.0571, 2012.

[CHSS13] Lorenzo Clemente, Frédéric Herbreteau, Amélie Stainer, and Grégoire Sutre. Reachability of communicating timed processes. In *Proceedings of the 16th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'13)*, volume 7794 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2013.

[CM06] Prakash Chandrasekaran and Madhavan Mukund. Matching scenarios with timing constraints. In *Proceedings of the 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 2006.

[CS08] Pierre Chambart and Philippe Schnoebelen. Mixing lossy and perfect fifo channels. In *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 340–355. Springer, 2008.

[dAHM03] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP'03)*, volume 2719 of *Lecture Notes in Computer Science*, pages 1022–1037. Springer, 2003.

[dLAMJM11] Wilkerson de L. Andrade, Patrícia D. L. Machado, Thierry Jéron, and Hervé Marchand. Abstracting time and data for conformance testing of real-time systems. In *Proceedings of the 4th International IEEE Conference on Software Testing, Verification and Validation (ICST'12)*, pages 9–17. IEEE, 2011.

[DLL+10] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed i/o automata: a complete specification theory for real-time systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC'10)*, pages 91–100. ACM, 2010.

[DLLN09] Alexandre David, Kim G. Larsen, Shuhao Li, and Brian Nielsen. Timed testing under partial observability. In *Proceedings of the 2nd International Conference on Software Testing Verification and Validation (ICST'09)*, pages 61–70. IEEE, 2009.

[DSZ10] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *Proceedings of the 21th International Conference on Concurrency Theory (CONCUR'10)*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2010.

[EM79] Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

[END03] Abdeslam En-Nouaary and Rachida Dssouli. A guided method for testing timed input output automata. In *Proceedings of the 15th IFIP International Conference on Testing of Communicating Systems (TestCom'03)*, volume 2644 of *Lecture Notes in Computer Science*, pages 211–225, 2003.

[Fin06] Olivier Finkel. Undecidable problems about timed automata. In *Proceedings of the 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.

[FJ13] John Fearnley and Marcin Jurdziński. Reachability in two-clock timed automata is PSPACE-complete. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP'13)*, volume 7966 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2013.

[GHKK05] Hermann Gruber, Markus Holzer, Astrid Kiehn, and Barbara König. On timed automata with discrete time - structural and language theoretical characterization. In *Proceedings of the 9th International Conference on Developments in Language Theory (DLT'05)*, volume 3572 of *Lecture Notes in Computer Science*, pages 272–283, 2005.

[GKM07] Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80(1-3):147–167, 2007.

[GMNK09] Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Reachability and boundedness in time-constrained MSC graphs. In *Perspectives in Concurrency Theory*, IARCS-Universities, pages 157–183. Universities Press, 2009.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[Han93] Hans-Michael Hanisch. Analysis of place/transition nets with timed-arcs and its application to batch process control. In *Proceedings of the 14nd International Conference*

on Application and Theory of Petri Nets (ICATPN'93), volume 691 of *Lecture Notes in Computer Science*, pages 282–299, 1993.

[HLMS10] Alexander Heußner, Jérôme Leroux, Anca Muscholl, and Grégoire Sutre. Reachability analysis of communicating pushdown systems. In *Proceedings of the 13th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'10)*, volume 6014 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010.

[IDP03] Oscar H. Ibarra, Zhe Dang, and Pierluigi San Pietro. Verification in loosely synchronous queue-connected discrete timed automata. *Theoretical Computer Science*, 290(3):1713–1735, 2003.

[IT11] ITU-T. Z.120: Message sequence charts (MSC). Technical report, International Telecommunication Union, 2011.

[JJ93] Thierry Jéron and Claude Jard. Testing for unboundedness of fifo channels. *Theoretical Computer Sciences*, 113(1):93–117, 1993.

[JJ05] Claude Jard and Thierry Jéron. TGV: theory, principles and algorithms. *Software Tools for Technology Transfer*, 7(4):297–315, 2005.

[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.

[KCL98] O. Koné, R. Castanet, and P. Laurencot. On the fly test generation for real time protocols. In *Proceedings of the 7th International Conference on Computer Communications & Networks (IC3N'98)*, pages 378–387. IEEE, 1998.

[KJM04] Ahmed Khoumsi, Thierry Jéron, and Hervé Marchand. Test cases generation for nondeterministic real-time systems. In *Proceedings of the 3rd International Workshop on Formal Approaches to Software Testing (FATES'03)*, volume 2931 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2004.

[KL07] Orna Kupferman and Yoad Lustig. Lattice automata. In *Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'07)*, volume 4349 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2007.

[KNSS02] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.

[KT04] Moez Krichen and Stavros Tripakis. Black-box conformance testing for real-time systems. In *Proceedings of the 11th International SPIN Workshop on Model Checking Software (SPIN'04)*, volume 2989 of *Lecture Notes in Computer Science*, pages 109–126. Springer, 2004.

[KT09] Moez Krichen and Stavros Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, 2009.

[KY06] Pavel Krcál and Wang Yi. Communicating timed automata: The more synchronous, the more difficult to verify. In *Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 249–262. Springer, 2006.

[Lip76] Richard J. Lipton. *The Reachability Problem Requires Exponential Space*. Department of Computer Science, Yale University, 1976.

[LMN05] Kim Guldstrand Larsen, Marius Mikucionis, and Brian Nielsen. Online testing of real-time systems using Uppaal. In *Proceedings of the 4th International Workshop on Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *Lecture Notes in Computer Science*, pages 79–94. Springer, 2005.

[LMS04] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2004.

[LS98] Xinxin Liu and Scott A. Smolka. Simple linear-time algorithms for minimal fixed points. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 53–66. Springer, 1998.

[Mar11] Nicolas Markey. Robustness in real-time systems. In *Proceedings of the 6th IEEE International Symposium on Industrial Embedded Systems (SIES'11)*, pages 28–34. IEEE, 2011.

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

[MF85] Gérard Memmi and Alain Finkel. An introduction to fifo nets-monogeneous nets: A subclass of fifo nets. *Theoretical Computer Science*, 35:191–214, 1985.

[Min67] Marvin Lee Minsky. *Computation: finite and infinite machines*. Prentice-Hall, 1967.

[MK10] Lakshmi Manasa and Shankara Narayanan Krishna. Integer reset timed automata: Clock reduction and determinizability. CoRR arXiv:1001.1215v1, 2010.

[Mol82] Michael K. Molloy. Performance analysis using stochastic Petri nets. *IEEE transactions on Computers*, 100(9):913–917, 1982.

[NS03] Brian Nielsen and Arne Skou. Automated test generation from timed automata. *Software Tools for Technology Transfer*, 5(1):59–77, 2003.

[OW04] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS'04)*, pages 54–63. IEEE, 2004.

[OW05] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *Proceedings of the 20th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 188–197. IEEE, 2005.

[Pac82] Jan K. Pachl. Reachability problems for communicating finite state machines. Research Report CS-82-12, University of Waterloo, May 1982.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE, 1977.

[PP92] Wuxu Peng and S. Purushothaman. Analysis of a class of communicating finite state machines. *Acta Informatica*, 29(6/7):499–522, 1992.

[Pur00] Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

[Rac78] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

[Ram74] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974. Project MAC Report MAC-TR-120.

[Rei08] Klaus Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008.

[SBMR13] Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust controller synthesis in timed automata. In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 546–560. Springer, 2013.

[Sch61] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and control*, 4(2):245–270, 1961.

[SPKM08] P. Vijay Suman, Paritosh K. Pandya, Shankara Narayanan Krishna, and Lakshmi Manasa. Timed automata with integer resets: Language inclusion and expressiveness. In *Proceedings of the 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008.

[ST08] Julien Schmaltz and Jan Tretmans. On conformance testing for timed systems. In *Proceedings of the 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2008.

[Sta12] Amélie Stainer. Frequencies in forgetful timed automata. In *Proceedings of the 10th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'12)*, volume 7595 of *Lecture Notes in Computer Science*, pages 236–251. Springer, 2012.

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.

[Tri06] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. *Information Processing Letters*, 99(6):222–226, 2006.

[UDL] UPPAAL DBM library, `http://people.cs.aau.dk/~adavid/UDBM/python.html`.

[UL10] Mark Utting and Bruno Legeard. *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2010.

[vB78] Gregor von Bochmann. Finite state description of communication protocols. *Computer Networks*, 2:361–372, 1978.

[ZHM97] Hong Zhu, Patrick A. Hall, and John H. R. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, 1997.

Key Word Index

(Σ_1, Σ_2)-refinement, 66, 69
$\mathsf{Aut}(\sigma)$, 45, 47, 61, 69
$\mathcal{G}_{\mathcal{A},(k,M')}$, 42, 47, 58, 61
$\mathcal{G}_{\mathcal{A},(k,M')}$, 58, 68, 69
tioco, 86, 88, 89
ε-closure, 58, 68
ε-transition, 57, 79

Abstract valuation, 120, 121
Acyclic, 174, 191
Aperiodic, 127, 128, 156, 157, 159, 160
Approximate Determinization, 36, 52
Approximate determinization, 36, 37, 65, 92
Atomic reset, 48

Büchi semantics, 117

Channel, 174
Co-reachable, 81
Communicating counter automata, 181, 182
Communicating tick automata, 176, 179, 182, 184, 185, 187, 191, 196
Communicating timed automata, 175, 176, 189, 191, 196
Communicating timed processes, 174, 175
Communication actions, 174
Compatible, 82
Complementable, 35
Complete, 28, 82
Conformance relation, 86, 88, 89
Conformance testing, 37, 85
Contraction, 132, 133, 136, 140, 141, 143, 144
Corner, 119
Corner-point abstraction, 119, 121–129, 131–157, 159–162
Cost, 122
Counter automata, 181, 184, 185

Delay domain, 173
Deterministic, 29, 82, 86
Determinizable, 30, 82
Determinization, 35–37, 54, 92
Diagonal constraint, 32
Dilatation, 132, 133, 141, 144

Emptiness problem, 116, 159
Emptiness test, 173, 174, 177, 182, 184, 185, 187, 196–198
Equivalent, 29, 82, 175, 191
Event-clock timed automata, 36
Exhaustive, 87, 98

Forgetful, 126–129, 147–157, 159, 160, 162
Frequency, 114, 116, 117, 119–129, 131–157, 159–162
Frequency-based languages, 116

Guard, 27, 28

Implementation, 85
Input-complete, 82, 85, 87
Integer reset timed automata, 36, 56
Invariant, 27, 28, 59, 67, 79
io-abstraction, 93
io-refinement, 88–90, 93

Labeled transition system, 173
Language, 29, 30

Mimicking, 121, 137, 139, 154–156
Mixed, 149, 150

Non-blocking, 82, 85, 87
Non-determinizable timed automata, 35
Non-Zeno, 137, 140
non-Zeno, 30, 145

Observed clock, 79, 90

Open timed automaton with inputs and outputs, 79
OTAIO, 79

Parallel product, 83
Path, 173
Petri net, 184
Pointed region, 119
Polyforest, 174, 184, 185, 187, 196
Polytree, 174
Precise, 91, 97
Prefix, 28
Product, 84, 92
Projection, 31, 121–129, 131–157, 175
Proper clock, 79, 90
Prototype, 71
Python, 71

Ratio, 122
Reachability problem, 28, 175, 179, 184, 185, 187, 196
Reachable, 81
Region, 30
Region abstraction, 191
Region automaton, 31, 32
Region equivalence, 30
Relation, 41
Repeatedly observable, 98
Rescheduling lemma, 192–196
Reward, 122
Reward-converging, 123, 139, 143, 144, 149, 150
Reward-diverging, 137, 140, 145
Run, 173

Semantics of timed automata, 28
Sequence, 81, 85
Skeleton, 52, 53
Sound, 87, 88, 90, 97
Star, 185
Strategy, 45, 47, 69
Strict, 87, 97
Strongly forgetful, 126, 149, 155–157
Strongly non-Zeno, 30, 36, 152, 159, 160, 162
Syntax of timed automata, 27

TAIO, 80
Test case, 86, 93, 97
Test execution, 100
Test generation, 91
Test purpose, 90
Test suite, 86, 97
Test-free, 174, 191, 196
Testable, 174
Timed automata, 176
Timed process, 174
Timed word, 29, 30
Topology, 174
Trace, 81, 83, 173

Under-approximation, 65
Universality problem, 116, 160
Update of relations, 42, 59
Urgency, 177

Valuation, 27
Verdict, 86, 95

Weak timed simulation, 29

Zeno, 30, 138, 139, 141, 144, 161, 162
Zero test, 182, 184, 185
Zone, 71

Bibliographic Index

[AAC12], 169
[ABG+08], 25, 169
[ABG07], 169
[ACHH92], 20
[AD09a], 23, 107, 113
[AD09b], 23, 107, 113
[AD10], 23, 107, 113
[AD90], 20, 27
[AD94], 20, 21, 27, 35, 39, 77, 161, 170, 189
[ADOW05], 21
[AFH94], 22, 36, 53
[AHKV98], 66
[AMPS98], 22, 30, 36
[ATP01], 23, 107
[BA11], 24, 110, 113, 126, 129, 130, 146, 163
[BB05], 77
[BBB+08], 23, 107, 111
[BBBB09], 22, 36, 39, 40, 52–56, 64, 75, 103, 203
[BBBM08], 23
[BBBS11], 24
[BBL08], 23, 24, 107, 109–111, 113, 118, 119, 124, 137, 146, 147, 163
[BCD05], 37, 39, 40, 47
[BD91], 20
[BDFP04], 49
[BDL+12], 35
[BFH+01], 23, 107
[BGP96], 21
[BJSK11], 23
[BJSK12], 23, 40
[BSJK11], 79
[BSJK11a], 22, 40
[BSJK11b], 40
[BSJK12], 79
[BZ83], 25, 167, 179, 181, 187
[Boc78], 25, 167
[Bou09], 32, 115
[CCH+05], 108
[CDE+10], 108, 109, 124
[CDH08], 108
[CE91], 21
[CF05], 168
[CFI96], 168
[CHR02], 110, 126
[CHSS12], 182, 185
[CHSS13], 25
[CM06], 168
[CS08], 168
[DLL+10], 66, 78
[DLLN09], 78, 100
[DSZ10], 169
[EM79], 108
[END03], 101
[Fin06], 22, 35, 39, 47
[GHKK05], 170
[GKM07], 168
[GMN08], 20
[GTW02], 45, 47
[HLMS10], 168, 178, 183
[Han93], 20, 168
[IDP03], 169
[IT11], 20, 168
[JJ93], 168
[JLL77], 20
[JT04], 78, 90
[K09], 78
[KCL98], 101
[KJM03], 23, 78
[KL07], 108
[KNSS02], 23, 107
[KT04], 77
[KT09], 22, 23, 36, 38–40, 52, 53, 64–66, 75, 77, 78, 85, 86, 100, 103, 203

[KY06], 25, 169, 170, 173, 174, 177, 180, 196–199
[LMN04], 77, 86
[LS10], 56
[LS98], 72
[MF85], 168
[Mar11], 24
[Mer74], 20, 168
[Min67], 181
[Mol82], 107
[NS03], 23, 77, 78
[OW04], 21, 161
[OW05], 162
[PP92], 168
[Pac82], 25, 178, 181, 187
[Pnu77], 21
[Pur00], 126
[Ram74], 20, 168
[SBMR13], 111, 163
[SPKM08], 22, 36, 56
[ST08], 77, 86
[Sch61], 108
[Sta12], 24
[Tre96], 85, 86
[Tri06], 22, 35, 39, 47
[UDL10], 72
[ZHM97], 90, 101
[dAHM03], 108